Tesis de Licenciatura

Modelos de percolación con enlaces eventualmente abiertos a distancia unitaria y a distancia fija k con k arbitrario

---

Departamento de Computación

Facultad de Ciencias Exactas y Naturales

Universidad de Buenos Aires

Tesista
Guillermo Amaral

Director
Pablo Miguel Jacovkis

Codirectora
Mariela Sued

12 de marzo de 2009

Resumen

En este trabajo describimos el diseño, la implementación y las simulaciones numéricas de fenómenos de percolación para los cuales aún no se conoce solución teórica, y que no han sido suficientemente estudiados numéricamente. Concretamente, la percolación anisotrópica (las probabilidades de que los enlaces estén abiertos no son todas iguales sino que dependen de la dirección de cada enlace) y la percolación truncada o finita de largo alcance. Para los primeros estudiamos el alcance del cluster expandido a partir de un sitio dado (como el origen), mientras que para los segundos analizamos el comportamiento cuando existen enlaces entre sitios a distancia unitaria y a distancia k, con k arbitrario. En particular, estudiamos la probabilidad de percolación a medida que k aumenta, cuando las probabilidades de primeros vecinos y nuevos enlaces se mantienen fijas.

Para la construcción del simulador proponemos un ambiente dinámico de objetos como *Smalltalk* ya que nos permite realizar un diseño de alto nivel, lo que resulta propicio para la exploración y desarrollo de los modelos estudiados. Además, nos brinda la flexibilidad y poder expresivo necesarios para la rápida implementación de conceptos básicos como grafos, patrones de conexión, isotropía, anisotropía, etc., así como su combinación en diferentes modelos de percolación, permitiendo al mismo tiempo refinar estos modelos y optimizarlos computacionalmente.

Abstract

In this work we describe the preparation, design and implementation of numerical simulations of percolation phenomena for which a theoretical solution is as yet unknown, and which have not been sufficiently studied numerically; namely, anisotropic percolation (the probabilities of edges being open are not the same but depend on the direction of each edge) and truncated or finite long-range percolation. In the first case we study the scope of a cluster from one specific site (such as the origin), while in the second case we analize the behavior when there are edges at distance 1 and at distance k, where k is an arbitrary integer greater than 1. In particular, the critical probability as k increases and the probabilities for first neighbors (i.e., at distance 1) and k-distance edges remain the same.

We construct the simulator using a dynamic object environment like *Smalltalk* since it allows us to make a high-level design which is suitable for the exploration and development of the models studied. This environment provides us with the flexibility and expressiveness needed to quickly model basic concepts such as graphs, connection patterns, isotropy and anisotropy, etc., and combining them in the different percolation models, while at the same time it enables us to refine these models and optimize computational times.

Agradecimientos

Este trabajo simboliza el final de un camino que me eriqueció tanto a nivel académico como personal, ya que además de conocimientos me encontré con personas formidables. Con ellas compartí estupendas tardes de estudio, largas jornadas de trabajo, minutos de tensión en la previa de los exámenes, momentos de alegría por las metas conquistadas y de bajón por algún tropiezo. Pero por sobre todas las cosas aprendí de ellos y con ellos. Quiero agradecer en particular los autodenominados «Los galácticos»: Ale, Chelo y Vinson. También a Marce, Herni, Arik y Claudio.

Quiero agradecer especialmente a mi novia Guadalupe quien paralelamente está por obtener su merecida licenciatura en «ciencias de la paciencia», por haberme sabido apoyar en innumerables momentos en los años más importantes de mi carrera.

Agradezco a mis directores Pablo y Mariela, quienes además de buenas ideas tuvieron una excelente predisposición. Como resultado, además de este trabajo, me quedan muchas ganas de trabajar en futuros proyectos junto a ellos.

Índice

1. Introducción **1**
1.1. El modelo . . . 1
1.2. Aplicaciones . . . 2

2. Elementos de teoría de percolación **6**
2.1. Definiciones básicas . . . 6
2.2. Tipos de modelos . . . 7
2.3. El fenómeno crítico . . . 9

3. Aspectos de la simulación **11**
3.1. Consideraciones sobre el modelo . . . 11
3.2. Algoritmo básico . . . 12
3.2.1. Construir el modelo . . . 13
3.2.2. Generar números pseudo-aleatorios para simular probabilidades . . . 13
3.2.3. Buscar un cluster percolante . . . 14
3.2.4. Análisis de complejidad . . . 15
3.2.5. BFS vs. DFS . . . 16
3.3. Representación computacional de grafos . . . 16
3.3.1. Representación por adyacencias . . . 16
3.4. Representación de vértices . . . 19
3.5. Representación de adyacencias: Patrones. . . . 20
3.5.1. Construcción de adyacencias . . . 22
3.5.2. Resolución dinámica de adyacencias . . . 24
3.5.3. Solución de compromiso . . . 26
3.6. La simulación como proceso . . . 26
3.7. Algoritmo para acotar p_c . . . 28

4. Diseño del simulador **31**
4.1. Objetos y *Smalltalk* . . . 31
4.1.1. Smalltalk . . . 32
4.2. Objetos principales . . . 32
4.2.1. *PercolationModel* . . . 32
4.2.2. *LatticeGraph* . . . 33
4.2.3. *Lattice* . . . 33
4.2.4. *AdjacencySolver* . . . 34
4.2.5. *PSMatrix* . . . 34
4.2.6. *GraphPattern* . . . 35
4.2.7. *OpenPolicy* . . . 35
4.2.8. *ModelSampler* . . . 36
4.2.9. *PercolationPathFinder* . . . 37
4.2.10. *GraphSearchAlgorithm* . . . 38
4.2.11. Las variables: *ModelVariable* . . . 38
4.3. Objetos gráficos . . . 39

4.3.1. *PSChartObject* 39
4.3.2. *PSDrawer* 39
4.3.3. *PSDrawerTool* 40
4.4. Diseño de la interfaz gráfica de usuario (GUI) 40
4.4.1. Patrones 40
4.4.2. Critical Range Finder 42
4.4.3. Simple simulation 45
4.4.4. Iterative simulation 47

5. Simulaciones **49**
5.1. Modelos isotrópicos en $\mathbb{L}^2$ 50
5.1.1. Square 51
5.1.2. Bow-tie 54
5.1.3. Hexagonal 57
5.1.4. Triangular 59
5.2. Modelos anisotrópicos en $\mathbb{L}^2$ 60
5.2.1. Square Vertical/Horizontal 62
5.2.2. Análisis de alcance 65
5.3. Modelos isotrópicos con enlaces a distancia 1 y a distancia k en $\mathbb{L}^2$ 72
5.3.1. Notación $Square : (V : \{\ldots\}, H : \{\ldots\})$ 72
5.3.2. Conjetura 73
5.3.3. $Square(V : \{1, k\}, H : \{1, k\})$ 74
5.3.4. $Square(V : \{1, k\}, H : \{1\})$ 75
5.3.5. $Square(V : \{1\}, H : \{1, k\})$ 76
5.3.6. $Square(V : \{k\}, H : \{1\})$ 77
5.3.7. $Square(V : \{1\}, H : \{k\})$ 78
5.4. Modelos isotrópicos en $\mathbb{L}^3$ 79
5.4.1. Cube 80

6. Conclusiones **81**

7. Bibliografía **83**

1. Introducción

La teoría de los procesos de percolación nació en la década del '50 cuando Broadbent y Hammersley plantearon un modelo probabilístico para estudiar el flujo de un fluido a través de un medio poroso aleatorio. El experimento que motivó este modelo era el siguiente: suponiendo la inmersión de una piedra porosa en un recipiente con agua resulta de interés estudiar la probabilidad de que el agua llegue hasta el centro de la piedra.

Desde entonces ha surgido mucho interés en el estudio de estos modelos debido a diversas razones. En primer lugar, su gran simplicidad frente a los resultados cualitativamente realistas para una gran variedad de sistemas desordenados. En segundo lugar, como marco de investigación de modelos más complejos y de herramientas matemáticas que permitan obtener mayor conocimiento de los medios aleatorios estudiados. En este sentido, modelos aleatorios físicos complejos o modelos en medios desordenados pueden ser abordados mediante el estudio de modelos más simples y mejor conocidos (y por ende, más manejables) permitiendo derivar resultados parciales u obtener indicios sobre los modelos más complejos. Finalmente, existe un atractivo matemático en la resolución de una gran cantidad de conjeturas simples de formular pero de las cuales aún no se ha obtenido una demostración formal.

1.1. El modelo

El modelo planteado por Broadbent y Hammersley en dos dimensiones es el siguiente: sea $\mathbb{L}^2$ la retícula cuadrada definida sobre los puntos $x \in \mathbb{Z}^2$ con ejes $e = (x_1, x_2)$ entre los vecinos *más cercanos*, es decir, si la distancia en norma 1 de x_1 a x_2 es 1. O sea, $d(x_1, x_2) = |x_{11} - x_{21}| + |x_{12} - x_{22}| = 1$, donde $x_1 = (x_{11}, x_{12})$ y $x_2 = (x_{21}, x_{22})$. Sea p un número tal que $0 \leq p \leq 1$. Se considera cada eje *abierto* con probabilidad p y *cerrado* con probabilidad $1 - p$, en forma independiente del resto de los ejes. Cada eje representa un pasadizo en el interior de la roca y la probabilidad p la proporción de pasadizos que permiten el pasaje del agua sobre el total de pasadizos de la roca. Al sumergir la roca en un recipiente con agua, un punto en el centro de la roca es alcanzado por el agua solo si existe un *camino* abierto desde la superficie de la roca hasta tal punto. Dado que se puede suponer *despreciable* el tamaño de los pasajes interiores de una roca frente a al tamaño total de la misma, es razonable pensar que la probabilidad de que un punto en el centro de la roca sea alacanzado por el agua se comporte de forma similar a la probabilidad de que exista un camino abierto de longitud infinita.

El estudio de los procesos de percolación esta centrado en las condiciones para la existencia caminos abiertos o, de forma más general, grupos o *clusters* abiertos, de tamaño infinito.

1.2. Aplicaciones

Estas son algunas de las aplicaciones del modelo.

Incendios en bosques. La propagación de incendios en bosques puede ser modelada utilizando percolación. En el modelo más simple, el bosque está representado por una grilla cuadrada en la que sus vértices estan o bien ocupados por un arbol o bien estan desocupados. Más precisamente, cada vértice puede estar en uno de tres estados posibles: (a) *no quemado*, (b) *quemándose*, o (c) *sin arbol* (ya se quemó o nunca existió). La propagación del fuego puede ser simulada de la siguiente forma: en primer lugar, el punto donde se origina el fuego puede ser elegido en forma arbitraria o bien en forma aleatoria (para modelar por ejemplo los incendios producidos por luz solar). Una vez que un arbol se está quemando, existe una probabilidad p de que el fuego se propague a un arbol vecino. Tan rápido como el fuego se propaga a otro arbol, el estado de tal arbol cambia a *quemándose* (si es que no se estaba quemando ya) y dicho arbol puede ahora propagar el fuego a otro vecino. La simulación es iniciada entonces estableciendo ciertos sitios en la grilla en estado *quemándose* y siguiendo un procedimiento iterativo que verifica qué otros árboles se van quemando. Modificando las condiciones que son usadas para decidir si un arbol propaga el fuego a sus vecinos (probabilidad p) uno puede modelar diferentes características estáticas como la distancia entre los árboles, el tipo de combustible que representan, la elevación y pendiente del terreno, así como características dinámicas como la dirección del viento o la humedad del aire y la temperatura. Un *cluster* es en este caso un conjunto de árboles que se quemaron o se estan quemando.
Referencias sobre algunos de estos modelos pueden encontrarse en [NTD/00] y [CFA/00].

Propagación de epidemias. Con el mismo modelo descripto para modelar la propagación de incendios en bosques podemos modelar la propagación de una epidemia en un grupo de personas (la población de una ciudad, de un país, o de todo el mundo). De la misma forma que en el caso anterior podemos incorporar en el modelo las diferentes carácteristicas de la enfermedad, tanto estáticas como dinámicas. Observemos que también puede ser de interes describir el hecho de que una persona infectada con la enfermedad, luego de recuperada, haya generado anticuerpos y no pueda volver a ser infectada. Esta observación conduce a la extensión del modelo para incluir el paso del tiempo, en el sentido de que un vértice representando a una persona, puede volver al estado inicial. Ejemplos de estos modelos pueden encontrarse en [ZCL/01] y [MNE/00].

Redes eléctricas desordenandas. Si bien se puede calcular fácilmente la resistencia eléctrica de un bloque de un material cualquiera, resulta ser bastante más difícil calcular la resistencia de una mezcla de dos materiales. Mas aún si la mezcla es *desordenada*.

Consideremos un bloque compuesto por una mezcla desordenada de partículas de dos materiales A y B, donde A es un material conductor y B es un aislante perfecto. Al ser la mezcla desordenada es razonable asumir que cada partícula del bloque puede ser aleatoreamente del material A o del material B, independientemente del resto de las partículas.

Los agrupamientos de partículas del material A conducirán electricidad. Podemos aplicar un diferencial de potencia en dos caras opuestas del bloque para medir la resistencia efectiva del mismo. Si existe algún agrupamiento de partículas del material A que conecte ambas caras entonces la resistencia será finita, de lo contrario podemos considerar la resistencia infinita. La formación de tales agrupamientos dependerá de la proporción p de partículas del material A.

Veamos esto gráficamente. Sea U_n la sección $\{0, 1, \ldots, n\} \times \{0, 1, \ldots, n\}$ en $\mathbb{Z}^2$, y sean S_n y I_n los lados superior e inferior de U_n, respectivamente,

$$S_n = \{(m, 0) : 0 \leq m \leq n\}$$

$$I_n = \{(m, n) : 0 \leq m \leq n\}$$

Podemos interpretar a U_n como una red eléctrica en el siguiente sentido. Consideramos cada eje como un cable con resistencia 1 Ohm con probabilidad p, y ningún cable con probabilidad $1 - p$, en forma independiente del resto de los ejes. Luego consideramos I_n y S_n como si fueran barras de plata a las que aplicamos un diferencial de potencia (ver figura 1). Nos preguntamos *cuál es la resistencia efectiva R_n de la red*.

Como mencionamos el valor R_n depende de la densidad y geometría de los ejes que tienen resistencia unitaria. Se demuestra que para n lo suficientemente grande $R = \infty$ si $p < \frac{1}{2}$, mientras que $0 < R_n < \infty$ si $p > \frac{1}{2}$.

Existencia de petróleo o gas. La teoría de la percolación puede ser utilizada como una modelo simple para la predicción de la distribución del petróleo (o gas) en rocas porosas o reservorios.

En el modelo de percolación, podemos representar a la roca como una red de poros llenos de petróleo, donde cada poro es un vértice. Dado que en la roca los poros pueden estar conectados o pueden estar aislados, podemos representar este aspecto con ejes entre los vértice. La porosidad o el promedio de concentración de petróleo en la roca puede ser representada con la probabilidad con la cual un vértice es ocupado.

Uno de los factores deseables para maximizar la producción obtenida de un pozo, es que el mismo esté ubicado en un area con alta porosidad. Por tal razón, es importante poder estimar la porosidad de la roca en el area donde se asume que el reservorio está ubicado. Con una buena estimación de la porosidad es posible predecir la producción que se extraerá del reservorio y de esta se puede evaluar si la inversión será redituable.

Para realizar una estimación de porosidad se toman muestras de la roca. La dificultad radica en el hecho de que las muestras son usualmente pedazos de

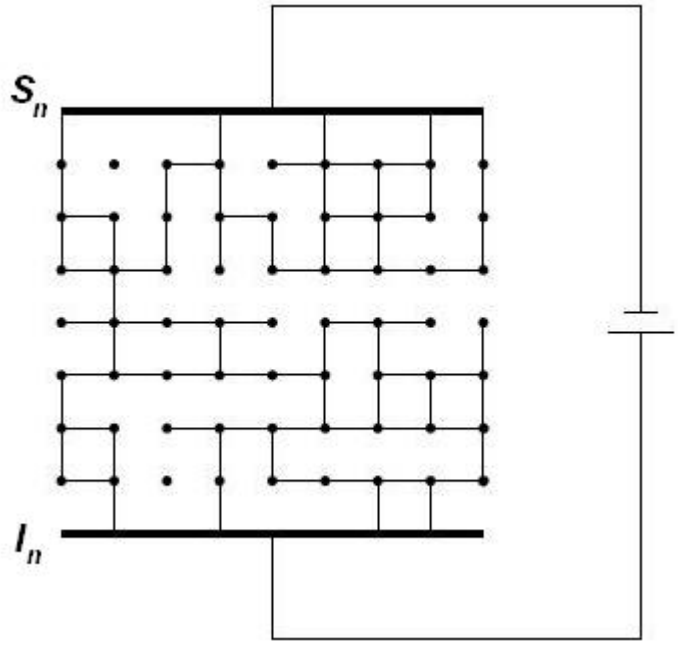

Figura 1: Modelo de una red eléctrica.

roca cuyo diámetro es del orden de los centímetros, y por lo tanto, es necesario extrapolar la medición de las muestras a la escala del reservorio (normalmente del orden de los kilómetros).

La pregunta fundamental es *cuándo dicha exptrapolación es legítima*. El enfoque desde la percolación dice que tal extrapolación es válida cuando la probabilidad p está notablemente por encima de la probabilidad crítica. Por otro lado, si la probabilidad está cerca del la probabilidad crítica, la decisión de invertir puede llegar a no ser redituable.

Pueden encontrarse referencias sobre estos modelos en [BRO/93] y [GPA/94].

Ferromagnetismo. Uno de los fenómenos críticos más estudiados en la física teórica es el del ferromagneto. Ponemos un trozo de un metal apropiado dentro de un campo magnético y observamos la forma en la cual la magnetización del metal varía de acuerdo a las oscilaciones en le campo externo. Supongamos que incrementamos el campo externo desde 0 a algun valor dado, y luego lo decrementamos hasta 0 nuevamente. Si la temperatura es suficientemente alta, el metal no retiene niguna magnetización residual, mientras que a bajas temperaturas el metal mantiene parte de la magnetización inducida. Exsite un valor crítico T_c de la temperatura, llamado el *punto Curie*, marcando el límite entre la existenca y la no existencia de la llamada *magnetización espontánea*. Un modelo matemático estandar para este fenómeno es el denominado *Ising model*.

En este modelo definido sobre la grilla $\mathbb{Z}^d$, cada vértice puede estar en uno de dos estados: 0 o 1, y se definen probabilidades condicionales sobre tales vértices. Tales probabilidades dependen de dos parámetros: el campo magnético externo h y la fuerza J de interacción entre los vecinos. Si $J = 0$ entonces el estado de los diferentes vértices es independiente, y el proceso es equivalente a un modelo de percolación de tipo site.

La relación entre este modelo y uno de percolación de tipo bond es muy estrecha. Ambos estan vinculados por medio de un tipo de percolación generalizada denominado *random-cluster model*. A través del estudio de este último, se obtienen conclusiones válidas simultáneamente para la percolación y para el *Ising model*. Una referencia sobre este modelo puede encontrarse en [MTW/73].

Como mencionamos anteriormente, éstas son algunas de las aplicaciones de la teoría de la percolación, existiendo otras tantas. Adicionalmente, se pueden someter a prueba conjeturas propias sobre una problemática dada, motivando tal vez nuevas conjeturas.

2. Elementos de teoría de percolación

En esta sección presentamos algunas conceptos fundamentales de la teoría de percolación.

2.1. Definiciones básicas

A continuación enumeramos algunas definiciones básicas sobre grafos y más abajo, a modo de ejemplo, una definición formal de un modelo de percolación concreto.

Definicion 1. Un *grafo* es un par ordenado $G = (V, E)$, donde V es el conjunto de *vértices*, y $E \subset V \times V$ es el conjunto de *aristas* o *ejes*, que relacionan estos vértices. Si $(v_1, v_2) \in E$ se dice que v_1 y v_2 son *adyacentes* y que el eje (v_1, v_2) *incide* sobre v_1 y sobre v_2.

Definicion 2. Dado un vértice $v \in V$, el *grado* de v es el número de ejes que inciden sobre v.

Definicion 3. $G = (V, E)$ es un grafo *no dirigido* si E es una relación simétrica, es decir, si dados v_1 y v_2 en V, se cumple que $(v_1, v_2) \in E \Leftrightarrow (v_2, v_1) \in E$.

Definicion 4. Dados v y w en V, se dice que $v \in V$ está *conectado* con w, y se denota $v \leftrightarrow w$, si existe un camino $e_1, \dots e_n$ tal que $e_i = (v_i, v_{i+1}) \in E$ con $v_1 = v$ y $v_{n+1} = w$. La longitud del camino es n, es decir, el número de ejes que lo forman.

Definicion 5. Un grafo es *conexo* si para todo v y w en V, v está conectado con w.

Definicion 6. Dados $G = (V, E)$ y $W \subset V$, $W \neq \emptyset$. El *subgrafo inducido* por W es el grafo $G_W = (W, E_W)$ tal que $E_W = \{(v, w) \in E : v, w \in W\}$.

Definicion 7. Supongamos que cada eje puede estar en uno de dos estados: *abierto* ó *cerrado*. Denominamos *camino abierto* al camino $e_1, \dots e_n$ tal que e_i está abierto para todo i, con $1 \leq i \leq n$. Alternativamente, supongamos que cada vértice puede estar *abierto* ó *cerrado*. También denominamos *camino abierto* al camino determinado por la sucesión $v_1, v_2, \dots, v_n \in V$ tales v_i está abierto para todo i, con $1 \leq i \leq n$ y $e_i = (v_i, v_{i+1}) \in E$.

Definicion 8. Dado un vértice cualquiera $v \in V$, el *cluster abierto* a partir de v, denotado C_x, es el conjunto de todos los vértices w tales que v está conectado con w mediante un camino abierto.

Supongamos que queremos asignar probabilidades a los estados (abierto y cerrado) de los ejes[1] de los grafos con los que vamos a trabajar. En el caso de un solo eje esto sería sencillo: bastaría considerar una variable aleatoria y el experimento sería un experimento de Bernoulli de parámetro p, si (por ejemplo) p es la probabilidad de que el eje esté abierto y $1 - p$ cerrado. Con finitos ejes

[1] El caso de estados asociados a los vértices es análogo.

también sería simble: bastaría con finitas variables aleaorias independientes para cada eje. La extensión a grafos infinitos requiere de algunos tecnicismos adicionales y es la que damos la siguiente definición.

Definicion 9. Tomemos como espacio muestral $\Omega = \{0,1\}^E$, cuyos elementos $\omega \in \Omega$ denominamos configuraciones tales que $\omega = (\omega(e) : e \in E))$ y donde $\omega(e) = 1$ implica que e está *abierto* y $\omega(e) = 0$ que e está *cerrado*.
Tomamos $\mathcal{F} = \prod_{e\in E} \mathcal{P}(0,1)$ la σ-álgebra producto (generada por los cilindros finito dimensionales). La medida producto con densidad $0 \leq p \leq 1$ en $(\Omega, \mathcal{F})$ es la medida $\mathbb{P}_p = \prod_{e\in E} \mu_e$ donde μ_e es una medida de Bernoulli en $\{0,1\}$ dada por:

$$\mu_e(\omega(e) = 1) = p,\ \mu_e(\omega(e) = 0) = 1 - p$$

Entonces $(\Omega, \mathcal{F}, \mathbb{P}_p)$ es un espacio de probabilidad.

Definicion 10. Se dice que X es un proceso de percolación independiente de parámetro p si es una variable aleatoria tomando valores en $\Omega = \{0,1\}^E$, cuya distribución es $\mathbb{P}_p$ la medida producto con marginales $(p, 1-p)$.

Estas dos últimas definiciones se refieren al modelo isotrópico *square bond*, el cual es uno de los modelos más simples y estudiados en la teoría de la percolación. Las mismas pueden ser extendidas para otros modelos más complejos como los que veremos a continuación; para los cuales no daremos una definición formal.

2.2. Tipos de modelos

Existen dos grandes familias de modelos de percolación de acuerdo con la condición de apertura: *bond* en donde los ejes son los elementos que pueden estar abiertos o cerrados y *site* en donde los vértices son aquellos que tienen dicha propiedad. Se puede demostrar (aunque no lo haremos) que un modelo de tipo *bond* puede ser expresado como un modelo de tipo *site* y no vale la inversa ([GRI/99]), por lo que los segundos son más generales que los primeros.

Los modelos se pueden clasificar de acuerdo a otras características como:

- las **dimensiones**: *2-dimensiones*, *3-dimensiones*,..., *n-dimensiones*;
- la **estructura** o **topología** de sus ejes: *square*, *bow-tie*, *hexagonal*, *kagomé*, etc. (ver figuras 2, 3, 4 y 5);
- la **uniformidad** del parámetro p en distintas **direcciones**: *isotrópicos*, mismo p en todas las direcciones, o *anisotrópicos*, diferentes parámetros para diferentes direcciones.

Naturalmente estas características no son excluyentes unas de otras sino que pueden combinarse para definir el modelo que mejor se ajuste la problemática

estudiada. Por ejemplo, podemos hablar de un modelo de tipo *bond*, en 2-*dimensiones*, con la estructura *square* y con parámetros p_V en el sentido vertical y p_H en el sentido horizontal. Este modelo podría utilizarse para describir el fenómeno de la propagación de incendios en un bosque, donde los vértices representarían los árboles y las diferentes probabilidades podrían representar una distancia mayor en un sentido que en otro, o la influencia del viento.

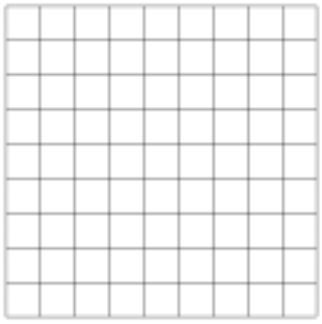

Figura 2: *Square*.

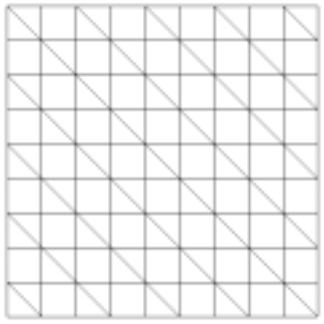

Figura 3: *Bow-tie*.

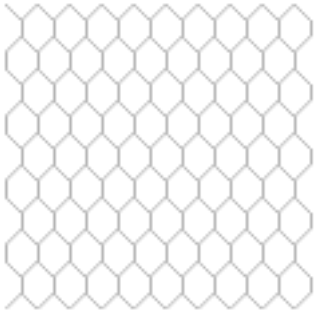

Figura 4: *Hexagonal*.

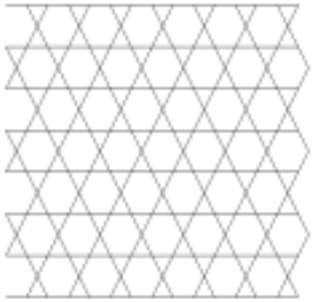

Figura 5: *Kagomé*.

2.3. El fenómeno crítico

Como mencionamos anteriormente el estudio de los procesos de percolación se centra en el estudio de la formación de clusters infinitos.

Uno de los factores que influyen en la existencia de clusters infinitos es naturalmente la probabilidad p [2].

Un fenómeno que se puede observar y que es bastante intuitivo es que si $p = 0$ entonces no existirá un cluster infinito ya que todos los ejes estarán cerrados (con probabilidad 1). Por otro lado, si $p = 1$ habrá un único clúster abierto ya que todos los ejes estarán abiertos (con probabilidad 1). Intuitivamente, a medida que p aumenta se formarán clusters de mayor tamaño (con mayor probabilidad).

Más precisamente, si llamamos $\theta(p)$ a la probabilidad de que un vértice dado pertenezca a un cluster infinito[3] en función del parámetro p tenemos que:

- $\theta(0) = 0$
- $\theta(1) = 1$
- θ es monónota no decreciente. [4]

Estas observaciones dan la pauta de la existencia de un valor de p, al que llamaremos *probabilidad crítica* o abreviadamente p_c, a partir de cual la probabilidad de que un vértice dado pertenezca a un cluster infinito es **estríctamente positiva**. Este fenómeno, deniminado *fenómeno crítico* es central en el estudio de los procesos de percolación y lo podemos resumir así:

$$\theta(p) \begin{cases} = 0 & \text{si } p < p_c, \\ > 0 & \text{si } p > p_c, \end{cases} \tag{1}$$

Formalmente se demuestra que para dimensiones mayores que 1, $0 < p_c < 1$ ([GRI/99]). Observemos que sobre la probabilidad crítica, es decir, $\theta(p_c)$ no afirmamos nada. Esta cuestión está aún abierta para muchos modelos, desconociéndose el comportamiento de $\theta(p)$ en tal punto.

El valor p_c depende de las características del modelo estudiadio como la topología de las conexiones (ejes), las dimensiones de la retícula, etc.

En la figura 6 podemos apreciar un gráfico de lo que se cree es la función $\theta(p)$.

Es particularmente llamativo el hecho de que se conocen muy pocos valores exactos [5] de p_c. Por ejemplo, el modelo *square* sobre $\mathbb{L}^2$ fue objeto de estudio durante 20 años, desde que Harris en 1960 postuló que $p_c \leq \frac{1}{2}$, hasta que Kesten en 1980 demostró que $p_c = \frac{1}{2}$ ([KES/80]).

[2] Aquí nos referimos a un modelo isotrópico, es decir, con una única probabilidad de apertura p.

[3] Esta función es equivalente para todo vértice ya que la probabilidad de que exista un cluster infinito es invariante por traslaciones en la grilla.

[4] Este hecho se demuestra formalmente aunque no incluiremos la demostración aquí. Teorema 2,1 en [GRI/99].

[5] Con valores exactos nos referimos a valores cuya validez reside en una demostración matemática rigurosa.

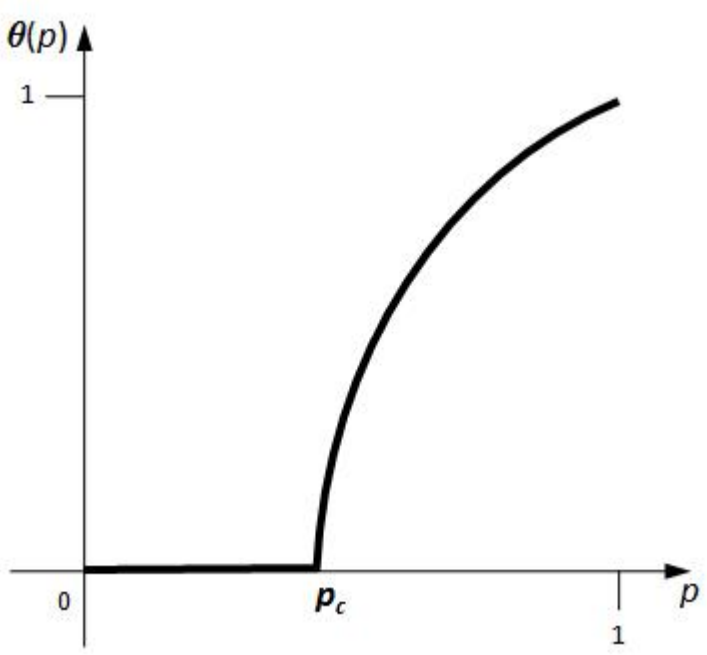

Figura 6: Función $\theta(p)$ *esperada*.

Es por esta razón que es de gran valor la utilización de simulaciones computacionales para analizar el comportamiento de diferentes modelos y poder acotar la región crítica de p, pudiendo en algunos casos orientar la búsqueda de pruebas formales del valor p_c.

A continuación podemos ver el cuadro 1 con algunas de las probabilidades críticas conocidas (demostradas formalmente).

	Bond	**Site**
Square	0,5	$\approx$ 0,59
Bow-tie	$1 - p - 6p2 - 6p3 - p5 = 0$	$\approx$
Hexagonal	$1 - 2sin(\pi/18)$	0,7
Triangular	$2sin(\pi/18)$	$\approx$
Kagomé	$\approx$ 0,52	$\approx$

Cuadro 1: Probabilidades críticas conocidas.

3. Aspectos de la simulación

Como hemos mencionado anteriormente, existen muchas conjeturas interesantes acerca de los procesos de percolación que encierran una gran dificulad a la hora de encontrar fundamentos matemáticos rigurosos que las validen o refuten. Esta dificultad da lugar a la utilización de simulaciones numéricas que refuercen las conjeturas o den indicios para una prueba formal. Al mismo tiempo, si se tiene una estimación de la *confianza* de los resultados de la simulación, se pueden utilizar tales resultados en la toma de decisiones sobre la situación real estudiada. En esta sección nos centramos en los principales aspectos que hacen a la simulación de modelos de percolación.

Inicialmente veamos algunas consideraciones importantes sobre el modelo a la hora de realizar una simulación.

3.1. Consideraciones sobre el modelo

Modelos infinitos vs. modelos finitos. La naturaleza de los modelos estudiados en la teoría de percolación es de caracter infinito, es decir, el tamaño de tales modelos es infinito. Cuando hablamos de simulación, nos referimos a modelos de tamaño finito, ya que la computadora es un medio esencialmente finito. Por este hecho, a la hora de estimar por ejemplo la función $\theta(p)$ mediante simulaciones, es necesario tener en cuenta que el tamaño del modelo puede influir en los resultados: cuanto mayor sea el tamaño del modelo simulado, más significativa será la simulación y por lo tanto mejor será la estimación de la función estudiada. Este tamaño, además de tener una cota *física* dada por la computadora, también tiene una cota *práctica*: idealmente buscamos que la simulación de un modelo se realice en un tiempo *aceptable*. Este último punto es más un juicio de valor por parte de quien diseña e implementa, y finalmente quien usa la simulación, que un criterio con fundamentos científicos. De todas formas, esta claro que no está dentro de lo aceptable que la simulación tome meses (o años) en arrojar un resultado (aún cuando este sea muy preciso).

Cluster percolante. Otro aspecto a tener en cuenta es que si bien el fenómeno crítico se basa en la existencia de un cluster infinito, en un modelo finito debemos reformular el fenómeno.

Una alternativa es definir un cluster *percolante* como con la propiedad de *conectar* extremos opuestos de la grilla mediante caminos compuestos de ejes (caso bond) o vértices (caso site) abiertos. Notemos que podemos remitirnos a la existencia de tal cluster en un solo sentido (horizontal ó vertical en el caso de 2 dimensiones; en 3 dimensiones contamos con otro sentido más), o bien podríamos considerar la existencia de un cluster percolante en cualquiera de los sentidos.

Otra alternativa es considerar condiciones de frontera periódicas y buscar un cluster que *envuelva* la grilla. Esta alternativa es un poco más difícícil de manejar que la de simplemente conectar extremos opuestos de la grilla.

Nuestro análisis se enfoca en la primer alternativa y en un único sentido: vertical.

También es importante mencionar que puede existir más de un cluster con tal propiedad[6] (ver figura 7). Este hecho es de suma importancia ya que a la hora de estimar la probabilidad de que en vértice pertenezca a un cluster percolante en función del parámetro p, es decir, el valor de la función θ, utilizaremos la proporción de vértices en **todos** los clusters percolantes respecto de la totalidad de los vértices del grafo.

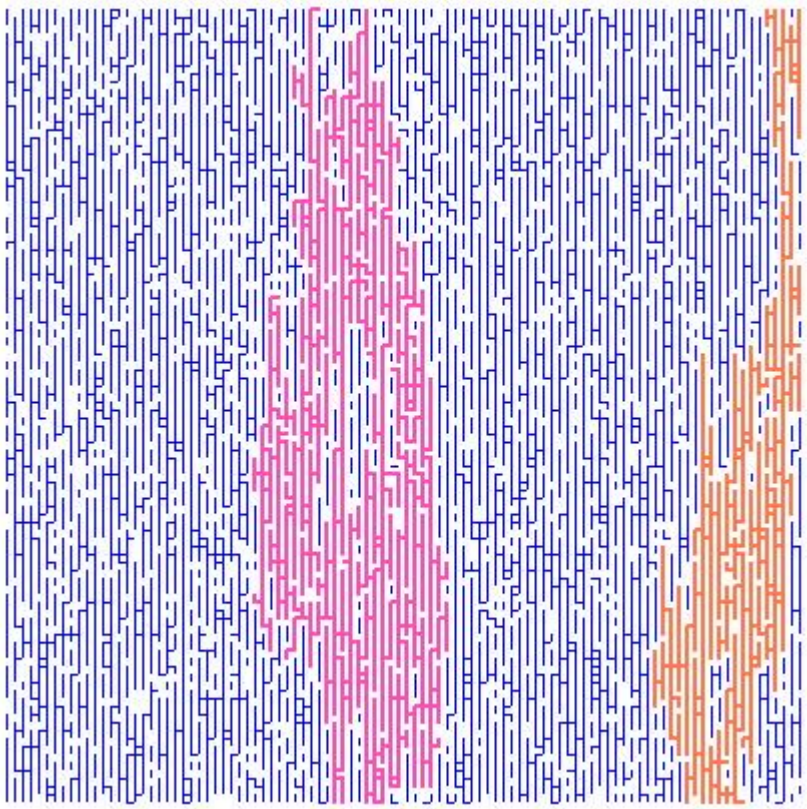

Figura 7: Más de un cluster percolante.

3.2. Algoritmo básico

La idea general de la simulación consiste en los siguientes tres pasos básicos:

1. Construir el modelo (grafo)

[6]Esto no se cumple en el caso de modelos infinitos ya que se demuestra que, de existir un cluster percolante, el mismo es único [GRI/99].

2. Generar números pseudo-aleatorios para simular probabilidades
3. Buscar un cluster percolante

3.2.1. Construir el modelo

Especificación del modelo. El punto de entrada de la simulación es la *especificación* del modelo a simular. Tal especificación consta de los siguientes elementos:

- **Tipo:** el tipo de modelo a simular, *Site* ó *Bond*.
- **Dimensiones:** de la grilla, 2 ó 3[7].
- **Tamaño:** *alto*×*ancho* en el caso de 2 dimensiones, y *alto*×*ancho*×*profundidad* en el caso de 3 dimensiones.
- **Patrón:** de conexiones del modelo (por ejemplo *square, bow-tie, hexagonal*, etc.).
- **Política de apertura:** *isotrópica* ó *anisotrópica*.
- **Probabilidad(es) de apertura:** dependiendo de la política elegida se deben especificar una o más probabilidades de apertura. Estos parámetros se utilizarán en la simulación para determinar si los ejes, en el caso de un modelo de tipo bond, o los vértices, en el caso de un modelo site, están abiertos o cerrados.

Dada esta especificación, la construcción del modelo consiste básicamente en la creación de un grafo para representar la grilla especificada. Es decir, sus vértices estarán asociados a los vértices de la grilla y sus ejes estarán determinados por el patrón especificado (ver en 3.5 para detalles sobre la forma de representar y construir este grafo).

Si utilizamos resolución dinámica de adyacencias entonces este paso no se realiza. De lo contrario, para construir el grafo (su lista de adyacencias) tendremos un orden de complejidad temporal de $O(|V| * G_{Max})$ como se explica en 3.5.1.

3.2.2. Generar números pseudo-aleatorios para simular probabilidades

Supongamos que estamos en el caso bond (el caso site es análogo solo que con vértices) y supongamos además que estamos frente a un modelo isotrópico (la generalización al caso anisotrópico es trivial). Dado que queremos simular el hecho de que un eje esta abierto con probabilidad p y cerrado con probabilidad $1 - p$, es decir, un *proceso de Bernoulli de parámetro p*, podemos proceder de la siguiente forma. Sea e un eje cualquiera:

[7]Estas son las dimensiones estudiadas en esta tesis y soportadas por el simulador.

- Generamos un número p_e pseudo-aleatorio en el intervalo $[0, 1]$,
- Comparamos dicho número con el parámetro p:
  - Si $p_e <= p$ entonces decimos que el eje e está abierto,
  - De lo contrario decimos que el eje e está cerrado.

Para que este procedimiento sea adecuado, la generación del número pseudo-aleatorio debe ser *uniforme* en el intervalo $[0, 1]$, de lo contrario no estaríamos modelando la situación descripta.

La problemática de generación de números pseudo-aleatorios ha sido ampliamente estudiada y existe una gran bibliografía al respecto. En nuestro caso no realizamos un análisis exhaustivo de diferentes métodos existentes sino que simplemente consideramos un método conocido como *Park-Miller* [PM/98] (una variante del método congruencial lineal), dejando tal análisis para un trabajo futuro.

La complejidad temporal de esta tarea es naturalmente $(|E| * c)$ en el caso bond, ya que debemos asignar un valor a cada eje, y $O(|V| * c)$ en el caso site, donde c es el costo de calcular un número pseudo-aleatorio y asignar dicho valor a un eje o vértice. Si estamos en el caso site, c corresponderá solo al costo de calcular un número pseudo-aleatorio ya que tales valores para cada vértice serán mantenidos en un *array* (de tamaño $|V|$), y su acceso tiene orden $O(1)$. Si estamos en el caso bond, y se utiliza la resolución dinámica de adyacencias entonces también será solo el costo de calcular un número pseudo-aleatorio, ya que los valores de los ejes serán mantenidos en un *array* (de tamaño $|E|$). Finalmente, en el caso de utilizar una lista de adyacencias, c corresponderá a la generación del un número pseudo-aleatorio y el costo de asignar un peso a un eje en la lista de adyacencias. Como se explica en 3.5 este último orden será $O(G_{Max})$, siendo G_{Max} el grado máximo del grafo.

3.2.3. Buscar un cluster percolante

Una vez que contamos con el grafo y el *estado de apertura* de los ejes o vértices, podemos proceder a verificar la existencia de un cluster percolante.

Recordemos que en nuestro caso, tal cluster conecta el extremo superior de la grilla con el extremo inferior. Dado que si existe tal cluster entonces en particular existe un camino abierto (contenido en el cluster) que une ambos extremos, podemos realizar una búsqueda de tal camino aplicando algún método para recorrer el grafo a partir de los vértices de un extremo. Si encontramos un vértice a partir del cual existe un camino abierto hasta algún vértice del otro extremo entonces existe un cluster percolante y decimos que el modelo *ha percolado*.

Nuevamente, existen diversos métodos para recorrer un grafo. Dos de los más elementales y de los denominados *ciegos* son *Breadth First Search* (BFS) y *Depth First Search* (DFS). Su clasificación se debe a que no basan su recorrido en ninguna información adicional como suele ser el caso de algunos métodos que *incorporan* heurísticas para la elección del recorrido a realizar.

DFS. Este método recorre el grafo primero *en profundidad*. A partir de un vértice dado, se aplica recursivamente el método en cada uno de sus vértices vecinos. De esta forma, expande todos y cada uno de los vértices visitados, de forma recurrente, y cuando no quedan más vértices que visitar, retoma el mismo procedimiento con los hermanos del último visitado. Se puede pensar gráficamente que este método recorre el grafo *rama por rama*. El orden de complejidad temporal de este método es $O(|V| + |E|)$ ya que cada vértice es visitado una vez y al igual que sus vecinos.

BFS. Este método recorre el grafo primero *en anchura*. A diferencia del método DFS, a partir de vértice dado visita en primer lugar **todos** sus vértices adyacentes. A continuación, para cada uno de dichos vecinos, explora sus respectivos vecinos adyacentes, y así sucesivamente hasta agotar todo el grafo. Se puede pensar gráficamente que este método recorre el grafo *nivel por nivel*. El orden de complejidad temporal es también $O(|V| + |E|)$ ya que cada vértice es visitado una vez y al igual que sus vecinos.

Como estamos interesados en buscar caminos abiertos, el método aplicado no debe tener en cuenta los vecinos propiamente dichos, es decir, aquellos determinados originalmente por el patrón del grafo, sino **solo aquellos vecinos unidos por ejes abiertos**. Pero esta condición se puede agregar fácilmente en cada paso del método elegido. Más precisamente, antes de considerar un vecino (cualquiera sea el método aplicado), se puede chequear el estado de apertura del eje que los une[8]. En caso de estar abierto el método sigue normalmente. De lo contrario, tal vecino no es tenido en cuenta y se continúa con el siguiente.

Al hallarse un camino abierto con este procedimiento se puede *expandir* el cluster a partir de cualquier vértice del mismo. En particular podemos elegir el vértice donde se comenzó el recorrido. Como mencionamos anteriormente es necesario contemplar la posiblidad de que existan varios clusters percolantes y por tal razón el procedimiento descripto no debe finalizar al encontrar el primer cluster sino continuar la búsqueda agotando todos los vértices de un extremo de la grilla.

La complejidad temporal de esta parte será $O(O_B * N_B)$ donde O_B es el orden de la búsqueda, en nuestro caso $O(|V| + |E|)$ tanto para DFS como para BFS, y N_B es la cantidad de vértices del extremo de la grilla donde se buscan caminos de percolación.

3.2.4. Análisis de complejidad

Teniendo en cuenta los órdenes de complejidad de los 3 pasos,

- Paso 1: $O(|V| * G_{Max})$,
- Paso 2: $(|E| * c)$ en el caso bond y $O(|V| * c)$ en el caso site,

[8]En el caso de site, es aún más simple ya que basta chequear el estado del vértice que se visita. Si está abierto entonces se prosigue. De lo contrario, se pasa al siquiente vértice.

- Paso 3: $O(|V| + |E| * N_B)$

podemos calcular el orden de complejidad del algoritmo como la suma de dichos órdenes.

3.2.5. BFS vs. DFS

Si bien ambos algoritmos tienen la misma complejidad temporal, a la hora de utilizarlos para un propósito específico, como buscar caminos de percolación de un extremo a otro en una grilla, tiene sentido realizar un análisis del desempeño de cada algoritmo.

Aprovechando el hecho de que el simulador permite graficar la curva de tiempo de ejecución de las simulaciones, realizamos una comparación entre ambos algoritmos y descubrimos que el algoritmo DFS tiene un mejor comportamiento que el algoritmo BFS. En la figura 8 podemos ver las curvas de tiempos (en milisegundos) de cada algoritmo, junto con un promedio de ambas. Si bien el modelo utilizado es un modelo ralativamente pequeño, alcanza para exponer la diferencia entre ambos algoritmos.

3.3. Representación computacional de grafos

A la hora de elegir una forma de representar un grafo en una computadora es importante evaluar su costo en términos de tiempo de cómputo y espacio de memoria requeridos, buscando un balance entre ambos factores.

En términos generales existen dos grandes enfoques posibles: representar las *incidencias* o representar las *adyacencias*. El primer enfoque esta centrado en los ejes del grafo mientras que el segundo se centra en los vértices.

En el primer caso se puede optar por una *lista de incidencia*, es decir, una lista de pares de vértices, o bien por una *matriz de incidencia*, donde cada posición de la matriz representa la incidencia de un eje (dado por la fila o columna) sobre un vértice (dado por la columna o fila).

En el caso de representar las adyacencias también se puede optar por una *lista de adyacencia*, es decir, una lista de pares de vértices y listas de vecinos, o bien por una *matriz de adyacencia* donde cada posición en la matriz representa la adyacencia entre los vértices representados por la fila y la columna.

Las operaciones que deseamos resolver con la utilización de la representación elegida son básicamente dos: obtener todos los vértices adyacentes a un vértice dado y obtener el peso asociado a un eje (determinado por los dos vértices sobre los que incide).

A continuación detallamos sólo las alternativas del segundo enfoque dado que es el que mejor se adaptó a la problemática tratada en esta tesis.

3.3.1. Representación por adyacencias

Matrices de adyacencia. Esta es una de las formas más utilizadas para representar grafos en una computadora.

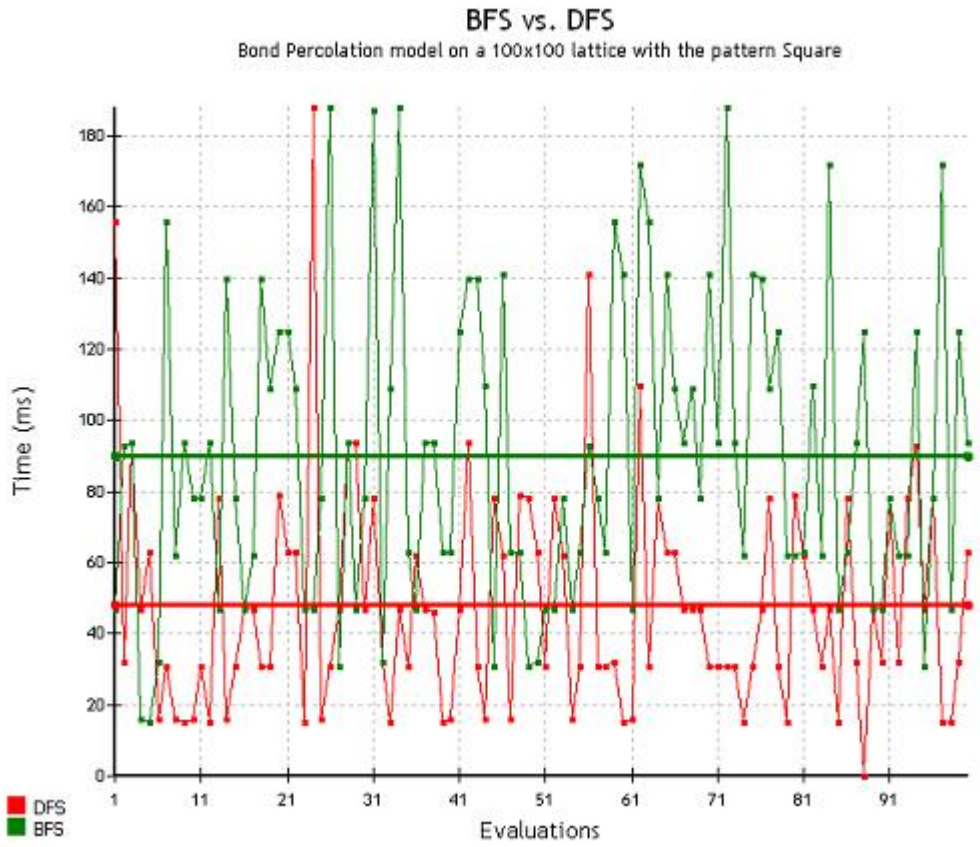

Figura 8: Tiempos de 100 simulaciones sobre un modelo *square* de 100×100.

Dado un grafo $G = (V, E)$, se define la matriz A de $n \times n$, con $n = |V|$, de forma tal que,

$$A[v_1, v_2] = \begin{cases} VERDADERO & \text{si } e = (v_1, v_2) \in E, \\ FALSO & \text{si no.} \end{cases} \tag{2}$$

Por lo tanto, el espacio de memoria requerido para representar un grafo es equivalente al espacio de memoria requerido para representar una matriz de $n \times n$.

Notemos que *VERDADERO* y *FALSO* pueden ser representados mediante cualquier valor, siendo los más intuitivos y mayormente utilizados 1 y 0. Esto permite implementar la matriz de adyacencia, por ejemplo, como una *matriz de bits*, optimizando el espacio de memoria requerido para su almacenamiento.

Se pueden realizar otras optimizaciones a nivel de espacio de memoria dependiendo de la particularidad de los grafos a representar. Por ejemplo, si se cumple que el grafo a representar tiene cierta topología o sus ejes cumplen

cierta propiedad. Supongamos que G es tal que, para todo $e = (v_1, v_2)$ en E, se cumple que $v_1 > v_2$; entonces la matriz A es claramente triangular y el espacio requerido es $\frac{n \times n}{2}$.

Otro caso que permite optimizar el espacio de memoria requerido es cuando el grafo a representar es *no dirigido* [9]. Recordemos que en un grafo $G = (V, E)$ no dirigido se cumple que para todo vértice v_1 y v_2 en V, si (v_1, v_2) pertenece a E entonces (v_2, v_1) también pertenece a E. Nuevamente, el espacio que se puede ahorrar al representar un grafo con esta propiedad es la mitad del espacio requerido para representar una matriz de $n \times n$. [10]

Frente a la necesidad de que los ejes de un grafo tengan valores o pesos asociados, una alternativa bastante intuitiva es almacenar dichos valores en la matriz de adyacencia. Supongamos que los pesos asociados son valores numéricos en $P \subseteq \mathbb{R}$. Dado un grafo $G = (V, E)$ y una función $f : E \to P$, se define la matriz de adyacencia como

$$A[v_1, v_2] = \begin{cases} f(e) & \text{si } e = (v_1, v_2) \in E, \\ NULL & \text{si no,} \end{cases} \tag{3}$$

El valor *NULL* se puede definir como un valor numérico arbitrario que no esté en P, o bien utilizar la convención adoptada por el lenguaje de programación utilizado.

Naturalmente, otra alternativa para representar un grafo con valores asociados a los ejes es tener dichos valores en una lista separada de la matriz de adyacencia. Esta alternativa tiene la desventaja de que se requiere de alguna función que mapee los ejes a los valores en la lista.

Respecto del tiempo de cómputo requerido para las operaciones que nos interesan, el acceso a los vecinos de un vértice tiene un orden lineal $O(n)$, con $n = |V|$, ya que basta con recorrer la fila asociada a un vértice. Por otro lado, acceder al peso asociado a un eje tiene orden constante, ya que dicho valor se encuentra en la posición dada por la fila y columna de los vértices adyacentes.

Listas de adyacencia. Dado un grafo $G = (V, E)$, se define la lista $L = \{p_1, ..., p_n\}$ de pares $p_i =< v_i, \{v_{i1}, ..., v_{ik}\} >$, donde $v_{i1}, ..., v_{ik}$ son los vértices adyacentes al vértice v_i (en cualquier orden).

Esta alternativa es conveniente para aquellos casos en donde las adyacencias son escasas, es decir, el grado de cada vértice es muy bajo en relación con la cantidad total de vértices del grafo, ya que optimiza el espacio de memoria requerido frente a la matriz de adyacencia.

Adicionalmente, se mejora levemente el tiempo de acceso a los vecinos debido al bajo grado de los vértices.

Observemos que en el caso de que los ejes tengan pesos asociados habría que extender esta representación para tal efecto. En este caso, y en relación con

[9] Observemos que la matriz de adyacencia sirve tanto para grafos dirigidos como para grafos no dirigidos.

[10] Naturalmente se deberán implementar los algoritmos de manipulación del grafo para que tengan en cuenta esta adopción.

el tiempo requerido para obtener el peso asociado a un eje, esta alternativa es menos eficiente ya que en el peor caso habría que recorrer todos los vecinos de uno de los vértices incidentes. [11]

3.4. Representación de vértices

Los modelos de percolación estudiados en esta tesis son modelos en retículas de dos y tres dimensiones. En ambos casos, resulta natural pensar a las retículas (puntos y enlaces) como grafos.

Dada una retícula o grilla, se puede ver que no es necesario almacenar el conjunto de vértices del grafo que la representa, ya que basta con definir alguna regla de enumeración de los mismos. Veamos gráficamente un ejemplo. Si estamos en presencia de una retícula de 3×3, podemos enumerar los vértices *de izquierda a derecha* y luego *de arriba hacia abajo* como se muestra en la figura 9.

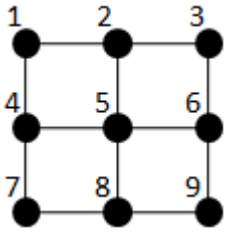

Figura 9: Ejemplo de numeración de nodos.

Otra alternativa podría ser enumerarlos *de arriba hacia abajo* y luego *de izquierda a derecha* (figura 10).

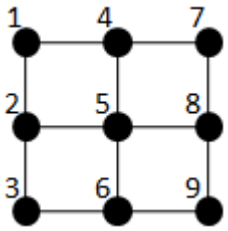

Figura 10: Ejemplo de numeración de nodos.

En el caso de modelos de tres dimensiones se puede proceder de forma análoga teniendo en cuenta una dirección más (por ejemplo, *de izquierda a derecha*, *de arriba hacia abajo y de adelante hacia atrás*).

[11] En un grafo no dirigido, si dos vértices son adyacentes, entonces ambos deben entar presentes en la lista de adyacencia de su vecino. En caso de tratarse de un grafo dirigido, a la hora de buscar el peso de un eje se deberá buscar en la lista de vecinos del vértice origen del eje.

Independientemente de la forma de enumeración adoptada, lo destacable es que no es necesario almacenar el conjunto de nodos sino que basta conocer la cantidad (dada por las dimensiones de la retícula) y su enumeración. Esta servirá para dar una *semántica* a los nodos y los ejes. Esto es, dada la primera forma de enumerar los nodos, los *nodos superiores* son los nodos 1, 2 y 3, y el eje que conecta los nodos 1 y 2 se interpreta como el eje *horizontal* que une el nodo superior izquierdo de la retícula con el inmediato a su derecha. De forma análoga, si adoptamos la segunda enumeración, los *nodos superiores* de la retícula son los nodos 1, 4 y 7, mientras que el eje que conecta los vértices 1 y 2 es el eje *vertical* que une el nodo superior izquierdo de la retícula con el inmediato inferior. Esta semántica servirá más adelante para definir los algoritmos de búsqueda de caminos de percolación así como también para describir las distintas políticas de apertura en los modelos anisotrópicos y naturalmente para poder implementar las herramientas de visualización.

3.5. Representación de adyacencias: Patrones.

Para aquellos modelos de percolación de tipo *site*, donde no se requieren valores en los ejes (ya que las probabilidades están asociadas a los nodos), el enfoque inicial y más natural de representar el grafo subyacente fue mediante una matriz de adyacencia implementada como una matriz de bits, mientras que, para los modelos de tipo *bond*, se utilizó una matriz de *punto flotante* para poder almacenar las probabilidades asociadas a cada eje. Ambas matrices en el fondo son listas[12] de bits y de *punto flotante*, respectivamente, de tamaño $n \times n$.

Estas representaciones resultaron convenientes y efectivas para los primeros modelos de prueba instanciados dado su reducido tamaño (del orden de las decenas), pero rápidamente se empezó a notar la ineficiencia de estos enfoques a la hora de instanciar modelos de mayor dimensión (del orden de las centenas), y la inviabilidad de representar modelos aún mayores. Por ejemplo, para representar un modelo de tipo *bond* sobre una retícula de 500×500, es decir, con 250000 nodos, y suponiendo que el tamaño de representación de un *punto flotante* es de 32 bits (4 bytes), el espacio de memoria requerido para la matriz de adyacencias es aproximadamente 232.8 GB ($\frac{250000 \times 250000 \times 4(bytes)}{2^{30}}$).

En un segundo acercamiento, y teniendo en cuenta que los grafos de los modelos estudiados tienen la característica de tener pocas conexiones por nodo, es decir, el grado máximo del grafo es muy bajo y no aumenta al aumentar las dimensiones de la retícula, fue utilizar una lista de adyacencia. Por ejemplo, el modelo denominado *bow-tie* tiene la estructura de la figura 11. Claramente se ve que en cada nodo inciden a lo sumo 6 ejes.

La primera alternativa estudiada dada su rápida implementación se basó en una estructura conocida como *diccionario*. Esta estructura básicamente se define como un conjunto de asociaciones $< clave, valor >$. En nuestro caso las claves serían los nodos y los valores serían asociaciones $< vecino, peso >$ (o sea, diccionarios con nodos como claves y los pesos como valores). Si bien esta

[12] Si bien conceptualmente son listas, utilizamos la clase *Array* en Smalltalk.

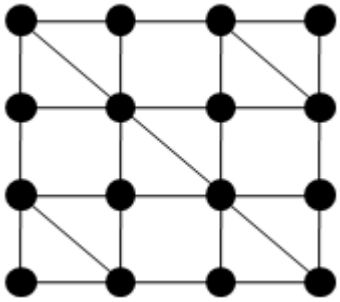

Figura 11: Grafo con la estructura *bow-tie*.

alternativa es considerablemente mejor que la anterior (respecto al espacio de memoria requerido) tiene un inconveniente: los diccionarios (en *Smalltalk*) se basan en *tablas de hash* para optimizar el acceso a su contenido en función de sus claves. Si bien esto es muy conveniente para no recorrer (en el peor caso) todos los vecinos de un nodo en busca de un vecino particular, estas tablas requieren de espacio adicional, el cual resulta condicionante para los casos estudiados, donde la cantidad de nodos, a pesar de su bajo grado, asciende al orden de los cientos de miles.

La siguiente implementación analizada fue una la lista de *punto flotante* de la siguiente forma. Supongamos que deseamos representar el grafo $G = (V, E)$ con $|V| = N$ sabiendo que el grado máximo del grafo es $k = G_{Max}$, con $k << N$; la lista de punto flotante tendría la estructura de la figura 12, donde $1 \leq v_{ij} \leq N$ y p_{ij} es el peso asociado al eje que va del nodo i y al nodo v_{ij}.

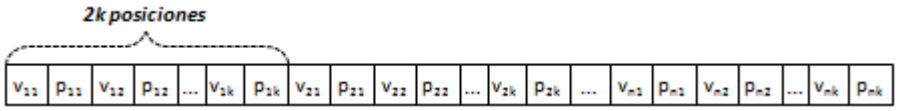

Figura 12: Estructura interna de la lista de adyacencia (como lista de punto flotante).

Por ejemplo, si k fuera 5 y el nodo 1 tuviera como vecinos a los nodos 3, 15 y 24 con los valores 0.2, 0.7 y 0.1 respectivamente, entonces tendríamos el contenido de la figura 13.

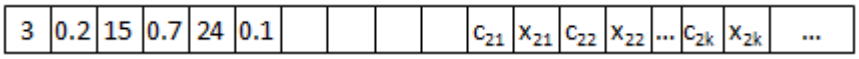

Figura 13: Ejemplo del contenido de una lista de adyacencia.

De esta forma, grafos con grandes cantidades de nodos pero con un grado muy bajo pueden ser representados optimizando el espacio de memoria requerido.

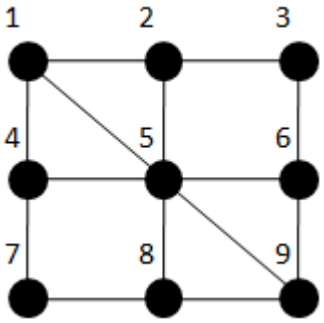

Figura 14: Patrón de la estructura *bow-tie*.

En cuanto al tiempo de acceso, tanto para obtener los vecinos de un vértice como para obtener el peso de un eje, ambos tienen un orden $O(G_{Max})$, ya que dado un nodo se requiere ubicar el *bloque* que contiene los vecinos del nodo dado (esto tiene un orden constante) y luego realizar una búsqueda en las G_{Max} posiciones del bloque.

3.5.1. Construcción de adyacencias

Naturalmente, una posibilidad frente a los diversos grafos de los modelos analizados era implementar algoritmos específicos para cada uno de ellos. Esta alternativa, además de no ser flexible, implicaba una gran cantidad de trabajo adicional.

Dado que los grafos de todos estos modelos presentan una estructura homogénea, en el sentido que sus conexiones respetan algún patrón, buscamos modelar este concepto para facilitar la construcción de las matrices de adyacencias y al mismo tiempo posibilitar la futura creación de otras estructuras de grafos que quisiéramos estudiar.

En un primer acercamiento, definimos un *patrón* como un subgrafo propio del grafo representado. De esta forma, sólo debíamos especificar la matriz de adyacencia del grafo patrón (de tamaño muy acotado) y luego podíamos implementar un algoritmo para el armado de la matriz de adyacencias del grafo general (de tamaño arbitrario) a partir del patrón.

Veamos un ejemplo. Sea G el grafo con la estructura *bow-tie* de la figura 11. Podemos observar que el patrón que se repite en dicho grafo es el que se muestra en la figura 14 junto con su matriz de adyacencias (cuadro 2) (adoptando la numeración de izquierda a derecha y de arriba hacia abajo).

Si quisiéramos armar la matriz de adyacencia de un grafo cuyos nodos estan representados por la retícula de 7×7, basta con *iterar* tantas veces como *quepa* el patrón sobre la retícula, y en cada iteración conectar los nodos de acuerdo con el patrón. Gráficamente puede pensarse al patrón como un *sello* que aplicamos sucesivamente en un grafo (de mayor tamaño) replicando las conexiones entre

	1	2	3	4	5	6	7	8	9
1		1		1	1				
2	1		1		1				
3		1				1			
4	1				1		1		
5	1	1		1		1		1	1
6			1		1				1
7				1				1	
8					1		1		1
9					1	1		1	

Cuadro 2: Matriz de adyacencia del subgrafo patrón *bow-tie*.

sus vértices sobre el grafo subyacente.

Para realizar esta tarea hay que tener en cuenta un detalle técnico que es el siguiente: en cada iteración, para identificar los nodos del patrón con los nodos del grafo (y así poder copiar las conexiones), es necesario *transformar* la numeración de los nodos del patrón a los nodos del grafo. Esto se logra simplemente sumando el desplazamiento desde el origen de la retícula (de coordenadas $(1, 1)$) a las coordenadas de cada uno de los nodos del patrón.

Por ejemplo, si estamos en la iteración número 6, el desplazamiento desde el origen es $(2, 4)$ ya que hemos *desplazado* el patrón 2 filas verticalmente y 4 columnas horizontalmente como se ve en la figura 15.

En esta iteración se identifican entonces los nodos $1, 2, 3, 4, 5, 6, 7, 8$ y 9 del patrón con los nodos $19, 20, 21, 26, 27, 28, 33, 34$ y 35 del grafo respectivamente y se copian las conexiones. Dado que en el patrón el nodo 1 está conectado con el nodo 2, en el grafo el nodo 19 estará conectado con el 20, y así para todas las conexiones.

El orden de complejidad temporal de este algoritmo es $O(C_P * |V_P| * G_{Max})$, donde C_P es la cantidad de veces que *cabe* el patrón en el grafo, V_P son los vértices del patrón y G_{Max} es el grado máximo del grafo resultante. Esta última consideración se debe al orden de complejidad temporal de acceso en una lista de adyacencias (ver 3.5), dado que en cada paso debemos conectar un vértice con un vecino y ello requiere de un accesso a la lista de adyacencias.

Supongamos que el grafo es de 2 dimensiones, el tamaño de la grilla es $N \times M$, y el tamaño de la grilla del patrón es $N_P \times M_P$. Como C_P se calcula en función del tamaño del patrón respecto del tamaño del grafo, tenemos lo siguiente:

$$C_P = \frac{N}{N_P} * \frac{M}{M_P}$$

Es decir, la cantidad de veces que *cabe* en ancho multiplicado por la cantidad de veces que *cabe* en alto[13].

[13]En rigor se considera $\lceil (N/N_P) \rceil * \lceil (M/M_P) \rceil$ ya que no necesariamente los lados del grafo son

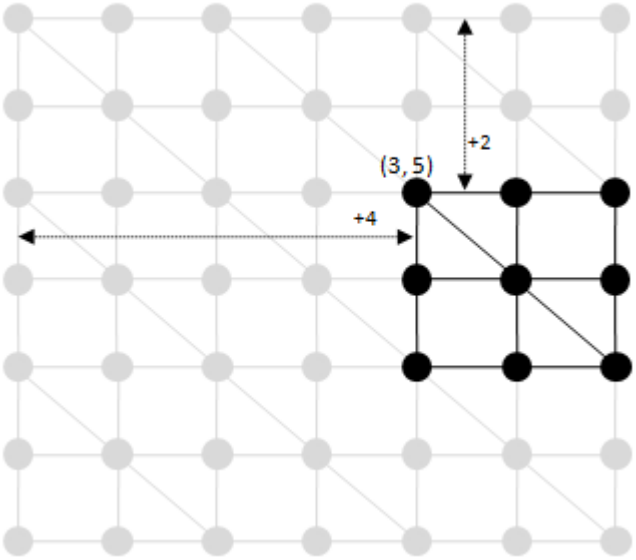

Figura 15: Desplazamiento del patrtón bow-tie en la retícula de 7×7 en la iteración número 6.

Podemos reemplazar este valor en la expresión anterior obteniendo:

$$O(\frac{N}{N_P} * \frac{M}{M_P} * |V_P| * G_{Max})$$

Dado que $|V_P| = N_P * M_P$ podemos reescribir lo anterior como

$$O(\frac{N}{N_P} * \frac{M}{M_P} * N_P * M_P * G_{Max})$$

Simplificando tenemos que:

$$O(N * M * G_{Max}).$$

Finalmente, como $|V| = N * M$ obtenemos que el orden es $O(|V| * G_{Max})$. En el caso de que el grafo sea de 3 dimensiones, con un procedimiento análogo, llegamos al mismo resultado.

3.5.2. Resolución dinámica de adyacencias

Dado que la información acerca de las conexiones del grafo que representa el modelo se concentra en el patrón, podemos utilizar dicha información en forma dinámica a la hora de resolver el vecindario de un vértice dado.

múltiplos de los lados del patrón. Sin embargo, para el cálculo del orden podemos obviar este detalle.

El procedimiento es básicamente el mismo empleado para la construcción de la lista de ayacencias: basta con *desplazar* el patrón en el grafo general de forma que *incluya* todos los vecinos del vértice dado. Sin embargo, hay que tener en cuenta que para agotar todos los vecinos puede ser necesario desplazar el patrón varias veces *alrededor* del vértice en cuestión. Veamos esto en un ejemplo.

Supongamos una grilla de 5×5 con el patrón bow-tie. Si quisiéramos resolver el conjunto de vértices adyacentes al vértice 15 deberíamos desplazar el patrón 4 veces, como se muestra en la figura 16.

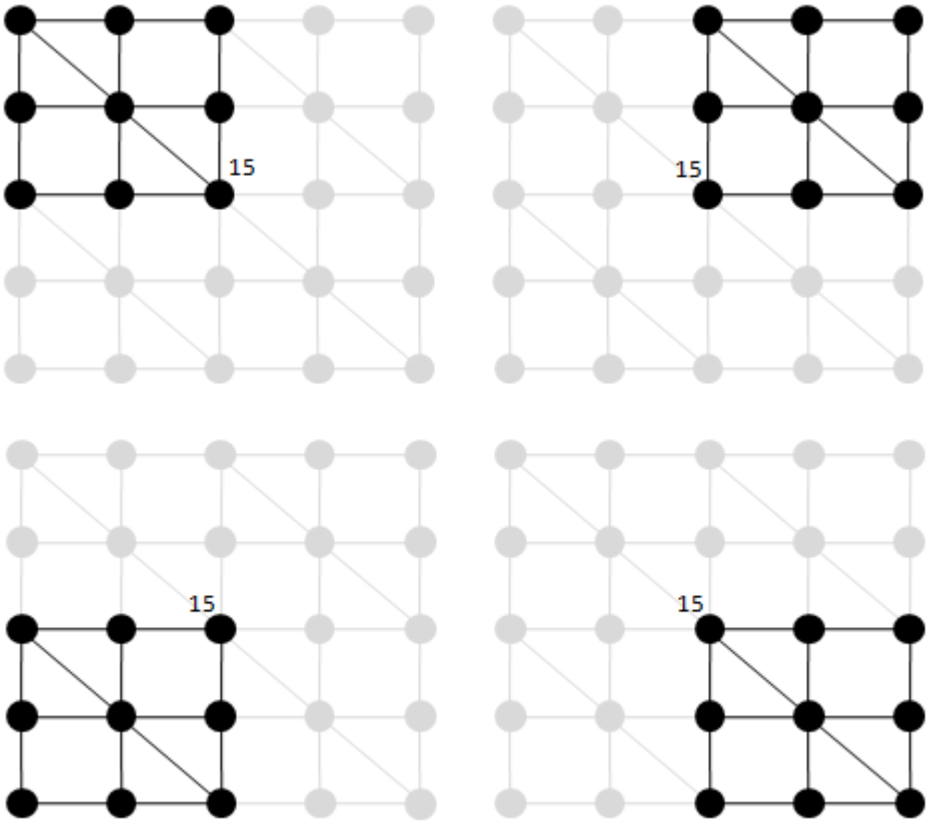

Figura 16: Desplazamiento del patrtón bow-tie en la retícula de 5×5 para resolver los vecinos del vértice 15.

Notemos que los vecinos determinados en cada desplazamiento no son necesariamente disjuntos ya que en el caso de dos desplazamientos contiguos, es decir, que *comparten* un eje, el vecino alcanzado por tal eje se repetirá en ambos desplazamientos (ver por ejemplo el primer y segundo desplazamiento en la figura). No obstante, se pueden descartar tales repeticiones.

De esta forma, es posible evitar el almacenamiento de las adyacencias reduciendo drásticamente la complejidad espacial de la representación del grafo

(solo se requiere espacio para almacenar las adyacencias del patrón).

Naturalmente, la complejidad temporal se ve afectada. En parte debido a los vecinos repetidos, ya que los mismos son contemplados más de una vez, y en parte porque es necesario realizar el cálculo de los desplazamientos para *abarcar* a todos los vecinos posibles.

3.5.3. Solución de compromiso

Teniendo en cuenta que para grafos relativamente pequeños (del orden de las centenas por lado) la complejidad espacial es aceptable, es preferible esta opción a la resolución dinámica de adyacencias. Por otra parte, si el tamaño del grafo es significativamente grande (del orden de los miles por lado) entonces podría comprometerse seriamente el almacenamiento de la lista de adyacencias, por lo que es conveniente la opción de la resolución dinámica de adyacencias.

Teniendo en cuenta estas observaciones podemos adoptar ambas opciones en forma dinámica dependiendo del tamaño del grafo. Esto es, definimos un umbral de tamaño por debajo del cual se utilizará una lista de adyacencias y por encima del cual se trabajará con resolución dinámica. El valor de este umbral requiere de un análisis detallado (que dejamos para un trabajo futuro).

3.6. La simulación como proceso

Recordemos que la simulación de un modelo específico consiste en:

1. Construir el modelo de acuerdo a la especificación,
2. Generar números pseudo-aleatorios, y
3. Buscar un cluster percolante.

En primer lugar notemos que el resultado de una simulación puede ser *éxito*, si existe un cluster percolante, o *fracaso*, en caso de que no exista tal cluster. Sin embargo, es posible *observar* otros resultados de la simulación, como por ejemplo el **tamaño del cluster percolante** (si lo hubiere), la **cantidad de ejes (o vértices) abiertos**, la **cantidad de ejes (o vértices) cerrados**, etc. Podemos pensar entonces a la simulación como **un proceso que parte de valores para un conjunto de *variables de entrada* y tiene como resultado valores para un conjunto de *variables de salida***. Ejemplos de variables de entrada son todos los parámetros de la especificación del modelo, como por ejemplo p (más de uno dependiendo de la política de apertura), $alto$, $ancho$, $profundidad$, etc.

Con el fin de analizar estadísticamente el comportamiento de un modelo podemos efectuar un número arbitrario de simulaciones o *evaluaciones* del mismo, cambiando (o no) los valores de las variables de entrada y recolectando los resultados sucesivos de las variables de salida.

Es importante notar que algunas variables de entrada afectan el resultado del paso 1 mientras que otras no. Por ejemplo, si varían las variables que determinan el tamaño de la grilla (*alto*, *ancho*, *profundidad*) entonces cambia la construcción del modelo, mientras que si si tales variables permanecen constantes y varía por ejemplo la variable p, la construcción del modelo no cambia. Esto es importante por cuestiones de optimización ya que una vez realizado el paso 1, podemos repetir suecivamente solo los pasos 2 y 3, *ahorrándonos* la construcción del modelo, si lo que nos interesa es analizar el impacto de la variación de una variable de entrada como p sobre el comportamiento del modelo, dejando las variables como *ancho*, *alto* y *profundidad* con valores constantes.

Veamos ahora una variable de salida que nos resultará de mucha utilidad.

Estimador de $\theta(p)$.

Fundamento teórico. El cálculo propuesto para el estimador de $\theta(p)$ está basado en los siguientes fundamentos. Recordemos que si C_x denota el cluster del punto x, tenemos que

$$\theta(p) = P_p\left(|C_0| = \infty\right) = E_p\left[I_{\{|C_0|=\infty\}}\right] ,$$

cualquiera sea x, donde P_p y E_p denotan probabilidades y esperanzas en el modelo donde las aristas están abiertas con probabilidad p, e I es la función indicadora. El Teorema ergódico garantiza que

$$\frac{1}{|\Lambda_n|}\sum_{x\in\Lambda_n} I_{\{|C_x|=\infty\}} \to E_p\left[I_{\{|C_0|=\infty\}}\right] .$$

Luego, como la sucesión de variables está acotada, la convergencia es preservada al tomar la esperanza. Tenemos entonces que

$$E\left[\frac{1}{|\Lambda_n|}\sum_{x\in\Lambda_n} I_{\{|C_x|=\infty\}}\right] \to \theta(p) .$$

Partiendo de este resultado teórico, reemplazamos la esperanza por un promedio y truncamos la suma por la suma en la caja finita. Por último, usando la unicidad de cluster infinito, consideramos que todos los clusters percolantes dentro de la caja finita forman parte del mismo cluster infinito.

Simulación. Por lo mencionado anteriormente, de existir un cluster percolante podemos estimar la probabilidad de que un vértice pertenezca al mismo como la proporción de los vértices en tal cluster percolante respecto de la totalidad de los vértices del grafo. En caso de existir varios clusters percolantes debemos utilizar la proporción de vértices en todos los clusters percolantes.

$$\widehat{\theta}(p) = \begin{cases} \frac{\#nodos(clusters\ percolantes)}{\#nodos(grafo)} & \text{si existe un cluster percolante,} \\ 0 & \text{si no existe un cluster percolante,} \end{cases} \tag{4}$$

Podemos entonces realizar un estudio del comportamiento de este estimador efectuando sucesivas simulaciones de nuestro modelo para diferentes valores del parámetro p.

Naturalmente, podemos realizar un promedio de un número arbitrario de evaluaciones para un mismo valor de p para obtener una mejor aproximación. Cuanto mayor sea el número de evaluaciones, mejor será la aproximación que obtengamos.

3.7. Algoritmo para acotar p_c

Siguiendo la idea de efectuar sucesivas simulaciones podemos definir un algoritmo para estimar una región o intervalo que contenga la probabilidad crítica p_c (con cierto margen de error).

Partiendo de un intervalo inicial, la idea básica consiste en incrementar el límite inferior (respectivamente decrementar el límite superior) si al cabo de un número arbitrario de evaluaciones el modelo no ha percolado (respectivamente ha percolado), con la variable p igual a dicho límite.

Observemos que estamos en el caso donde, una vez construido el modelo, podemos obviar este paso en las sucesivas evaluaciones, ya que la única variable de entrada que variaremos es p.

Parámetros del algoritmo. Llamemos I a la cantidad de veces que se incrementará (decrementará) el límite inferior (superior), E a la cantidad de evaluaciones en cada incremento (decremento) y A al porcentaje de aceptación sobre las E evaluaciones. Adicionalmente, llamemos p_{inf} y p_{sup} a los límites inferior y superior, e Inc y Dec al incremento y decremento de dichos límites, respectivamente.

Los parámetros I y E son valores enteros positivos. El parámetro A es un porcentaje expresado como fracción, es decir, $0 \leq A \leq 1$. Los parámetros p_{inf} y p_{sup} son probabilidades y satisfacen $0 \leq p_{inf} \leq p_{sup} \leq 1$. Finalmente, Inc y Dec, por ser *desplazamientos* de una probabilidad, deben cumplir que $0 < Inc, Dec < 1$.

Algoritmo. Los pasos del algoritmo son los siguientes:

- Repetir I veces:
  1. Efectuar E evaluaciones del modelo con $p = p_{inf} + Inc$.
     - Si A % de las E evaluaciones el modelo no ha percolado entonces
       - $p_{inf} = p_{inf} + Inc$
       - Inc queda igual.
     - De lo contrario,
       - p_{inf} queda igual
       - $Inc = \frac{Inc}{2}$.
  2. Efectuar E evaluaciones del modelo con $p = p_{sup} - Dec$.

- Si A % de las E evaluaciones el modelo no ha percolado entonces
  - $p_{sup} = p_{sup} - Dec$
  - Dec queda igual.
- De lo contrario,
  - p_{sup} queda igual
  - $Dec = \frac{Dec}{2}$.

Los pasos 1 y 2 no tienen dependencias entre sí y pueden realizarse en cualquier orden. Al cabo de las I iteraciones, el intervalo estará dado por $[p_{inf}, p_{sup}]$.

El parámetro A nos permite establecer el grado de *precisión* del algoritmo o, inversamente, el margen de error. No debemos dejar de lado que aquí no podemos hablar de precisión en el sentido numérico, aún en el caso de $A = 1$. En primer lugar porque estamos trabajando sobre modelos finitos y una variación en el tamaño puede ocasionar una variación en los resultados. Y en segundo término porque el generador de números pseudo-aleatorios podría introducir algún sesgo en las simulaciones.

En la figura 17 podemos ver un diagrama de flujo del algoritmo.

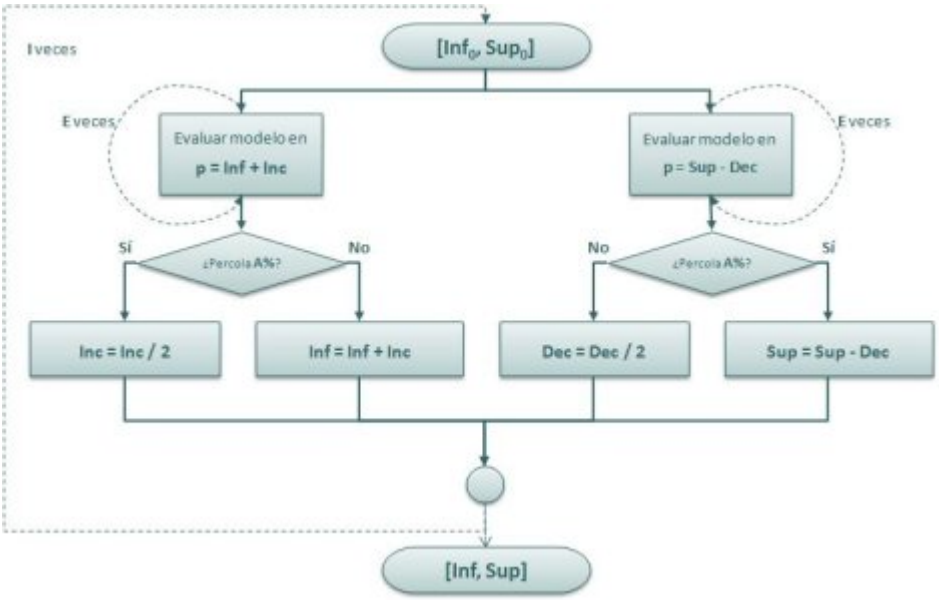

Figura 17: Algoritmo para acotar la probabilidad crítica.

Para tener mayor precisión es necesario simular modelos de tamaños considerablemente grandes (del orden de los cientos o miles de nodos por lado). Cuanto mayor sea el modelo, mayor precisión tendremos en los resultados de la simulación.

Naturalmente, si aumentamos la cantidad de evaluaciones (E) en cada iteración también obtendremos mejores resultados estadísticos.

Observaciones.

- El parámetro A da una pauta del *esfuerzo* para desplazar los límites: cuanto mayor es, más se exigirá para avanzar un límite. Obviamente este parámetro no debería estar por debajo de un cierto umbral ya que de lo contrario el resultado carecería de sentido. Un umbral aceptable puede ser 0,8.
- Esta claro que los valores iniciales de p_{inf} y p_{sup} cumplen que: $p_{inf} < p_{sup}$ (el caso $p_{inf} = p_{sup}$ carece de sentido).
- Si los valores iniciales son tales que $p_{inf} < p_c < p_{sup}$ entonces nunca debería pasar que $p_{sup} < p_{inf}$, es decir, que los límites se *cruzaran*. Como esto depende de factores como A (ie., si A es muy bajo entonces los límites podrían tender a cruzarse), el algoritmo fuerza este invariante, es decir, $p_{inf} \leq p_{sup}$ durante toda la ejecución del algoritmo.
- Los intervalos en cada iteración son no crecientes, es decir, p_{inf} nunca disminuye y p_{sup} nunca aumenta.
- De los tres puntos anteriores se sugiere que si p_c es menor que el valor inicial de p_{inf}, al finalizar el algoritmo, p_{inf} no se habrá desplazado y p_{sup} se habrá desplazado hacia p_{inf} lo máximo posible de acuerdo a I.
- Lo mismo ocurrirá si p_c es mayor que el valor inicial de p_{sup}. Al finalizar el algoritmo, p_{sup} no se habrá desplazado y p_{inf} se habrá desplazado hacia p_{sup} (nuevamente, lo máximo posible de acuerdo a I).
- Los parámetros *Inc* y *Dec* dan una idea de *cuán rápido* los límites se desplazan hacia el otro extremo. De acuerdo a la lógica del algoritmo, estos valores se ajustarán en forma dinámica, disminuyendo a medida que el algorimo avance. Es por esto que el valor inicial de estos parámetros debería estar en concordancia con el parámetro I: si iteraremos pocas veces no es conveniente comenzar con un valor muy pequeño de *Inc* y *Dec*, de lo contrario los límites se desplazarían poco hacia p_c y no obtendríamos un buen intervalo. Un valor inicial adecuado puede ser 0,1.

4. Diseño del simulador

En esta sección describimos los principales aspectos del diseño de un simulador para los modelos de percolación que queremos estudiar.

Aquí aparece la primer cuestión importante relacionada con nuestro objetivo. Decimos que buscamos diseñar un simulador *para los modelos de percolactión que nos interesa estudiar* y esto podría conducirnos a un simulador cuyo único alcance sea el estudio de tales modelos. Por el contrario, si buscamos un simulador que nos permita estudiar *cualquier* modelo, entonces quizas caigamos en un problema inabordable.

Sin embargo, existe un enfoque que nos permite buscar nuestro objetivo (un simulador que cumpla con el requerimiento de estudiar determinados modelos) sin descuidar que cierta generalidad es también deseable y posible, en el sentido de que podemos independizar nuestra implementación de los modelos concretos a estudiar.

Otra cuestión importante es la complejidad de nuestra implementación. Ya mencionamos en secciones previas la importancia de la *complejidad temporal* y la *complejidad espacial* del programa o algortimos que lo implementan. Sin embargo, existe además otra noción de complejidad y es la referida a la *complejidad del cógido* resultante. Nuevamente, podemos ocuparnos de que nuestro simulador cumpla con los requerimientos y tenga un desempeño aceptable en términos de tiempo de cómputo y espacio de memoria requerido, pero la comprensión de su código sea difícil e intrincada, o bien podemos buscar mantener la complejidad de código bajo cierto umbral, al igual que las otras complejidades mencionadas.

Estas dos cuestiones (independencia y simplicidad) apuntan a que el software resultante pueda *evolucionar* con el tiempo y sobre todo que permita (un objetivo esencial) enriquecer nuestra comprensión acerca la problematica estudiada y producir nuevos interrogantes. En definitiva, generar conocimiento. Con una fuerte dependencia del problema estudiado y una alta complejidad en la compresión de su código, nuestro software estaría condenado a ser útil momentáneamente, y nosotros estaríamos obligados a repetir el proceso de implementación al momento de estudiar nuevos modelos en un futuro.

4.1. Objetos y *Smalltalk*

Teniendo en cuenta los objetivos mencionados es que pensamos en un enfoque conceptual en lugar de un enfoque algorítmico. Es decir, buscamos *modelar* los conceptos que hacen a la simulación de procesos de percolación en lugar de *resolver* nuestro problema actual[14] mediante un conjunto de algoritmos. Al mismo tiempo, buscamos adoptar un enfoque *bottom-up*, es decir, representar los conceptos más básicos, para luego componerlos en estructuras más complejas.

[14]Con *problema actual* nos referimos al estudio de las características de algunos modelos de percolación, como es el caso de esta tesis.

Cuando hablamos de modelar nos referimos a representar computacionalmente un concepto, es decir, a crear un modelo virtual de dicho concepto, que nos permita interactuar y experimentar con él.

Notemos que existe una ambigüedad en el término *modelo* ya que podemos referirnos al modelo matemático de percolación, o bien al modelo computacional en el ambiente de objetos. Por este motivo, en esta sección aclaramos explícitamente cuándo nos referimos a uno o al otro.

4.1.1. Smalltalk

Una alternativa interesante para aplicar este enfoque conceptual es trabajar con objetos. Concretamente, elegimos *Smalltalk* como ambiente de objetos para trabajar y en particular, *Visual Smalltalk*, una implementación particular de *Smalltalk*, ya que contamos con alguna experiencia en su uso.

4.2. Objetos principales

A continuación damos una descripción de los principales objetos que componen nuestro diseño del simulador. Si bien realizamos varias iteraciones de diseño y desarrollo desde el primer prototipo, decidimos no describir cronológicamente los estados intermedios de cada iteración, con alguna excepción, en pos de dejar claro el diseño *final*[15].

4.2.1. *PercolationModel*

Esta es la principal abstracción en nuestro diseño y representa al modelo matemático de percolación. Es el marco que engloba los diferentes elementos que hacen a un modelo de percolación: el grafo (*LatticeGraph*), la política de apertura (*OpenPolicy*), y es responsable de determinar la apertura los ejes o vértices. Para este último propósito colabora con un generador de números pseudo-aleatorios (*RandomGenerator*).

Adicionalmente, permite especificar algunos aspectos de la evaluación como la dirección de búsqueda del camino de percolación (*Top to bottom*, *Left to right*, *Front to back*[16]), y el algoritmo utilizado, *BFS* o *DFS*. Para esto colabora con un *PercolationPathFinder* quien será el encargado de realizar la búsqueda (ver más abajo).

Para distinguir entre los dos grandes tipos de procesos de percolación, bond y site, creamos dos especializaciones: *BondPercolation* y *SitePercolation*. Básicamente, se diferencian en la forma de administrar la apertura de sus elementos, ejes en el primer caso y vértices en el segundo caso.

[15]Con *final* nos referimos al estado actual, ya que esperamos que el diseño siga evolucionando con el uso del simulador acompañado del descubrimiento de nuevas características del problema modelado.

[16]Solo para el caso de tres dimensiones

4.2.2. *LatticeGraph*

Como mencionamos, uno de los elementos constitutivos de un *PercolationModel* es un grafo. Tal grafo esta representado por un *LatticeGraph*[17].

Como cualquier grafo, está definido por el conjunto de sus vértices y por un conjunto de ejes entre sus vértices. Sin embargo, tiene la particularidad de que sus vértices estan determinados por una grilla (*Lattice*).

Su responsabilidad principal es dar el soporte para definir los ejes entre sus vértices, ya sea individualmente o bien a través de un patrón (*GraphPattern*), así como para examinar sus vértices y ejes. En el caso de utilizar un patrón, es capaz de *adoptar* tal patrón, es decir, construir su conjunto de adyacencias en función del patrón, o bien, dejar el conjunto de adyacencias *implícito*, determinando los ejes en forma dinámica. Ambas responsabilidades son llevadas a cabo en colaboración con un tipo particular de *AdjacencySolver*.

4.2.3. *Lattice*

Este objeto representa una grilla de puntos de coordenadas enteras.

Definimos dos tipos de grillas para representar la grilla en dos dimensiones (*SquareLattice*) y en tres dimensiones (*CubicLattice*).

Su responsabilidad es la de definir un mapeo entre la numeración de los vértices y un sistema de coordenadas. Recordemos que la numeración de los vértices es arbitraria. El sistema de coordenadas esta dado por las medidas de la grilla en las diferentes dimensiones, por ejemplo, *alto* y *ancho* en la grilla de dos dimensiones. Dado un vértice (su número), el mapeo consiste en determinar sus coordenadas, y viceversa.

Otra responsabilidad fundamental es la de definir qué patrones son válidos o compatibles, (por ejemplo, si la grilla es de dos dimensiones entonces los patrones válidos serán los de dos dimensiones), y de colaborar con un patrón dado para de definir las adyacencias. Esto es así, ya que tanto el proceso de adopción de un patrón como el de resolución dinámica de adyacencias, estan fuertemente vinculados con el manejo de las coordenadas de los vértices (para desplazar adecuadamente el patrón dentro de la grilla). Dependiendo de las dimensiones, cada grilla sabrá como colaborar con el patrón para tal tarea.

Adicionalmente, ofrece servicios para el acceso rápido a un determinado subconjunto de los vértices como *los superiores*, *los inferiores*, etc. Estos servicios son útiles para los objetos encargados de buscar caminos de percolación (i.e., caminos abiertos desde los nodos superiores a los nodos inferiores).

Es importante notar que este diseño nos permite extender fácilmente los tipos de grillas (proveyendo la implementación particular del protocolo común a todos los tipos). Y esto se puede hacer en forma transparente para el grafo ya que éste es independiente de las dimensiones de la grilla sobre la que se define.

[17] Su nombre proviene del conjunto de sus vértices: no es cualquier conjunto arbitrario sino el conjunto de vértices de una grilla (*Lattice*).

4.2.4. *AdjacencySolver*

Este objeto provee una interfaz homogénea para la resolución del conjunto de adyacencias de un grafo. Con *resolución* nos referimos a la definición y obtención de las adyacencias. Existen dos tipos de *AdjacencySolver*:

- *MatrixAdjacencySolver*: permite definir las ayacencias en forma individual (eje por eje) o global (dado un patrón). Como su nombre lo indica, mantiene esta información en una matriz de adyacencias[18] (*PSMatrix*). Para resolver las adyacencias colabora con tal matriz.
- *PatternAdjacencySolver*: permite definir las adyacencias en forma implícita a través de un patrón. Determina las adyacencias dinámicamente colaborando con el *GraphPattern* y la *Lattice*.

Es importante mencionar que decidimos no imponer la noción de grafo no dirigido, es decir, que con el mismo diseño podemos trabajar tanto con grafos no dirigidos como con grafos dirigidos. Esto puede resultar útil para futuros estudios de otros modelos de percolación.

4.2.5. *PSMatrix*

Una *PSMatrix*[19] representa una matriz de elementos de cualquier tipo. Su comportamiento básico consiste en brindar el acceso a sus elementos ya sea en forma individual (a traves de las coordenadas) o bien grupal (filas o columnas). También proveer diversas formas de iterar tales elementos.

Como comentamos en la sección 3.5, a lo largo del proceso de desarrollo probamos con diferentes implementaciones de matrices, comenzando con una implementación *naïve*, para luego refinar nuestro diseño y adaptarnos a los requerimientos de espacio de memoria requerido (recordemos que el almacenamiento de las adyacencias es un punto crucial en la reducción de la complejidad espacial). Cada uno de estos tipos fue modelado como un objeto diferente, pero con el mismo protocolo básico.

En el primer prototipo utilizamos una matriz simple, interpretada como una matriz de adyacencias (*PSMatrix*). Avanzando con la tarea de optimización, fuimos *descubriendo* nuevos tipos de matrices.

En primer lugar experimentamos con una matriz de adyacencias representada como una matriz de *bits* (*PSBitMatrix*).

Como decidimos aprovechar la matriz de adyacencias para alojar los valores aleatorios que determinan la apertura de los ejes (caso bond), modelamos una matriz de *punto flotante* (*PSFloatMatrix*).

Cuando decidimos cambiar el enfoque de matrices de adyacencia a listas de adyacencia, decidimos conservar la idea de matriz y modelamos el concepto

[18] También puede ser una lista de adyacencias, aunque conservamos el nombre debido a que tales listas están modeladas como matrices ralas.

[19] El prefijo *PS* se debe *PercolationSystem*, el nombre de la aplicación, y es sólo para distinguir estos nombres de otros ya existentes en el ambiente.

de *matriz rala* o *esparsa* (*PSSparseMatrix*). La diferencia respecto de las matrices anteriores es que esta optimiza el espacio de memoria, trabajando con listas de vecinos como explicamos anteriormente.

Finalmente, fusionamos la matriz esparsa y la matriz de punto flotante en una matriz con ambas características (*PSSparseFloatMatrix*).

4.2.6. *GraphPattern*

Este objeto representa un patrón de ejes en un grafo. Es muy importante haber modelado este concepto ya que independiza al grafo, así como al posterior análisis de caminos en tal grafo, de la definición de las adyacencias. De esta forma, una vez definidos los primeros, podemos extender nuestra gama de patrones en forma ilimitada, para estudiar nuevos modelos.

Su responsabilidad es definir las adyacencias de dos formas diferentes: a modo de subgrafo (*SubgraphPattern*), es decir, definiendo un grafo que se *propagará* por el grafo donde se aplique, o bien en forma de adyacencias de un vértice (*NeighborhoodPattern*). En este último caso, se define el vecindario de un vértice del grafo donde se aplicará el patrón (en términos de coordenadas).

Ambos tipos de patrón permiten definir familias de patrones no necesariamente disjuntas. Es decir, podemos definir el mismo grafo utilizando dos patrones, uno de cada tipo. Por ejemplo, el patrón *square* lo podemos definir utilizando un *SubgraphPattern* con cuatro vértices y cuatro ejes formando un cuadrado, o bien lo podemos representar con un *NeighborhoodPattern* que defina como vecinos de un vértice, de coordenadas (i, j), a los vértices de coordenadas $(i-1, j), (i+1, j), (i, j-1)$ y $(i, j+1)$.

Si bien los patrones de tipo subgrafo permiten definir una familia de grafos más extensa que los de tipo vecindario, por el simple hecho de que no imponen un mismo vecindario para todos los vértices, son más cómodos para definir patrones con alguna característica variable[20]. Un ejemplo de este tipo de patrones puede ser un *square* como el mencionado anteriormente, pero con los vecinos horizontales a una distancia k, con $1 \leq k$ variable.

4.2.7. *OpenPolicy*

Este objeto es el responsable de definir la política de apertura de un modelo de percolación: si el modelo es isotrópico, es decir, la misma probabilidad de apertura en todas las direcciones, o anisotrópico, o sea, diferentes probabilidades para diferentes direcciones.

En primer lugar notemos que para el caso de un modelo de percolación de tipo site, donde los vértices son los suceptibles de estar abiertos o cerrados, no tiene sentido pensar en diferentes probabilidades de apertura para diferentes direcciones, ya que un vértice no tiene dirección. Por esta razón, definimos la política *SiteOpenPolicy* como una política isotrópica: determina si un vértice está abierto o cerrado en función de una única probabilidad.

[20] Con *variable* nos referimos a una característica que no es fija en el patrón sino que puede variar.

Por otro lado, sí tiene sentido establecer diferentes probabilidaddes de apertura para diferentes direcciones en el caso de percolación de tipo bond, ya que un eje tiene una dirección (vertical, horitontal, etc.). Por tal motivo, definimos las políticas *IsotropicPolicy* y *AnisotropicPolicy*, como tipos especiales de *BondOpenPolicy*.

Una política isotrópica determina la apertura de un eje en función de una única probabilidad de apertura.

En el caso de una política anisotrópica hablamos de diferentes *direcciones*. Sin embargo, pensamos que nuestro diseño sería más flexible si pudiéramos definir diferentes probabilidades no solo en función de direcciones sino también en función de otras características. Por ejemplo, supongamos que queremos tener diferentes probabilidades de acuerdo con la *longitud* (medida en vértices) de un eje, es decir, si un eje une a dos vértices que se encuentran a una distancia k, con $k > 1$. Entonces, definimos un eje como abierto con una probabilidad p_k, mientras que si se encuentra a distancia unitaria utilizamos una probabilidad p_1.

Para nuestro trabajo experimentamos con tres políticas anisotrópicas:

- *VerticalHorizontalPolicy*: como su nombre lo indica esta política define dos probabilidades de apertura, una en el sentido vertical y otra en el sentido horizontal. Adicionalmente, decidimos agregar una tercer probabilidad, llamada *default*, para el caso de los ejes en otras direcciones, como por ejemplo diagonal.
- *OneKPolicy*: esta política es la del ejemplo mencionado anteriormente.
- *VerticalHorizontalFdPolicy*: esta política es igual que la primera pero agregaga una función de $f(d)$ (por eso Fd) para el cálculo de la probabilidad horizontal en base a la distancia d entre los vértices. Un ejemplo de función puede ser:

$$p_h(d) = \begin{cases} [d \cdot ln(d)]^{-1} & \text{si } d > 1, \\ 0 & \text{si no,} \end{cases} \tag{5}$$

4.2.8. *ModelSampler*

Estos objetos permiten realizar sucesivas evaluaciones de un modelo de percolación dado, manteniendo una historia de los resultados de cada evaluación. Para este último propósito colabora con una *ModelHistory*.

Su protocolo básico consiste en *correr*, *pausar* y *detener* una serie de evaluaciones de un *PercolationModel* dado. También ofrece servicios para realizar una ejecución *paso a paso*, deteniéndose luego de cada evaluación.

Existen diversos tipos de *ModelSampler* para diferentes propósitos. Estos varían en las acciones que efectúan en cada paso de la serie y son especializaciones del *ModelSampler*.

El más básico de todos es el *SimpleModelSampler*. Su responsabilidad consiste simplemente en evaluar el modelo, una cantidad específica de veces, y actualizar la historia en cada paso.

CriticalRangeFinder. Este es uno de los más útiles y es el encargado de llevar a cabo la búsqueda del intervalo o rango de la probabilidad crítica (ver 3.7). Para ello, en cada paso intenta incrementar el límite inferior y decrementar el límite superior. Una diferencia importante respecto del anterior es que la cantidad de evaluaciones no es arbitraria sino que está determinada por la cantidad de veces que se incrementará (decrementará) el límite inferior (superior) multiplicada por la cantidad de evaluaciones en cada incremento (decremento). Notemos que la historia de los resultados en las evaluaciones no es relevante en este caso, sino el resultado final: el intervalo más pequeño.

En algunos casos resulta conveniente efectuar una serie de evaluaciones con algun criterio, y en cada paso realizar otra serie de evaluaciones según otro criterio, por ejemplo, podemos modificar alguna característica del modelo a lo largo de la primer serie, y en cada paso evaluar varias veces el modelo para promediar los resultados.

Este es el comportamiento del *CompositeSampler*, un tipo especial de *ModelSampler*. Como su nombre lo indica es una *composición* de dos tipos de *ModelSampler*. A su vez, existen tipos específicos de *CompositeSampler* como el *VariableWalker*.

VariableWalker. Se encarga de evaluar un modelo sobre una suceción discreta de valores para una variable de entrada (*ModelVariable*), realizando un promedio de los resultados para cada uno de dichos valores. La sucesión de valores se define por comprensión con un valor inicial, un valor final y un incremento δ que determinará los valores intermedios. Un ejemplo de uso de este objeto es simular el comportamiento de la función $\theta(p)$, haciendo variar p de 0 a 1, con una granularidad dada por el incremento y una *suavidad* dada la cantidad de evaluaciones promediadas para cada valor de p.

NodeScopeCounter. Este objeto se caracteriza por no realizar una evaluación como el resto. En cambio, en cada paso realiza una expansión del cluster abierto a partir de un vértice dado, manteniendo un registro de los vértices alcanzados por tal cluster. Al cabo de una serie de pasos nos permite conocer la cantidad de veces que un vértice dado fue alcanzado.

En función de tales resultados podemos realizar diferentes análisis dependiendo de las carácterísticas del modelo estudiado (ver 5.2.2).

Existen otros tipos de *ModelSampler* y *CompositeSampler* para otros objetivos.

4.2.9. *PercolationPathFinder*

Este objeto representa el algoritmo de búsqueda de un camino de percolación. Permite definir la dirección del camino de percolación a buscar y el

algoritmo a utilizar (*GraphSearchAlgorithm*) para recorrer el grafo.

Recordemos que nuestro estudio se centró en la dirección vertical. Sin embargo agregamos la posibilidad de trabajar con diferentes direcciones: *Top to bottom*, *Left to right* y *Front to back*, para brindar una mayor flexibilidad.

El comportamiento es básicamente el explicado en 3.2.3.

Los vértices iniciales y finales estarán determinados por la dirección. Por ejemplo, si la dirección es *Top to bottom*, los vértices iniciales serán los superiores y los finales los inferiores.

4.2.10. *GraphSearchAlgorithm*

Este objeto representa un algoritmo de búsqueda sobre un grafo. Como mencionamos, decidimos utilizar dos algoritmos elementales: *BFS* y *DFS*, ambos representados por *BreadthFirstSearch* y *DepthFirstSearch* respectivamente, pudiendo definir nuevos algoritmos sin necesidad de modificar el resto del diseño.

Su comportamiento básico consiste en iterar sobre los vértices de un grafo a partir de un vértice dado. Adicionalmente permite establecer una condición de avance en la iteración. Esto es, antes de avanzar hacia un vértice la condición es evaluada. Si ésta se cumple, entonces avanza a tal vértice. De lo contrario sigue el recorrido por otro vértice. Culmina cuando no hayan más vértices por explorar.

La condición de avance es importante ya que es la utilizada para imponer la condición de apertura de ejes o vértices a la hora de buscar caminos de percolación.

4.2.11. Las variables: *ModelVariable*

Como mencionamos en la sección 3.6, podemos pensar en variables de entrada y de salida en el proceso de simulación de un modelo de percolación. Por esta razón, decidimos reflejar este concepto en nuestro diseño y lo representamos con el objeto *ModelVariable*.

Una *ModelVariable* básicamente consta de un nombre, un rango (*Range*) que determina los valores válidos para la variable y una marca que determina si la variable es de solo lectura, o de lectura/escritura. Esta última característica es utilizada para determinar si se trata de una variable de salida, es decir, que solo refleja un resultado (solo lectura), o bien es de entrada, permitiendo modificar su valor (lectura/escritura).

Las variables son utilizadas, por ejemplo, por los *ModelSampler*, quienes en cada evaluación actualizan la historia (*ModelHistory*) con los valores que toman las variables de salida (en dicha evaluación).

Otro ejemplo es el mencionado *VariableWalker*, quien en cada paso modifica el valor de una variable de entrada (determinada por el usuario), evalúa el modelo una cantidad de veces (también definida por el usuario), y calcula el promedio de los valores que tomaron las variables en dicha serie. Finalmente actualiza la historia de los valores promedio de todas las variables.

Es importante resaltar que en nuestro diseño **todos** los objetos pueden definir sus *propias* variables. Además, todo objeto ofrece un servicio para acceder a sus variables propias juntamente con las definidas por sus objetos dependientes, y las definidas por los dependientes de los dependientes, y así sucesivamente. Esto nos posibilita que a un nivel dado, podamos acceder a todas las variables relacionadas con dicho objeto.

Por ejemplo, un *PercolationModel* define variables como *Percolating Cluster Probability*, *Percolating Cluster Size* (que son de solo lectura). A su vez, la *OpenPolicy*, conocida por el *PercolationModel*, define variables como *Open Probability* (de lectura/escritura). Sin embargo, podemos utilizar el servicio provisto para acceder a todas las variables relacionadas con el *PercolationModel*: las propias y las de su *OpenPolicty*.

4.3. Objetos gráficos

Estos objetos llevan a cabo la tarea de graficación, tanto de los grafos asociados a los modelos de percolación, así como de los distintos tipos de gráficos que muestran los resultados de la simulación.

4.3.1. *PSChartObject*

Estos objetos representan los diferentes elementos que componen los gráficos de curvas y de tortas, que utilizamos para mostrar los resultados de la simulación. Algunos ejemplos son:

- *PSChart*: representa un gráfico con un conjunto de series. Definimos dos tipos de gráfico: de curvas *PSXYPlot* y de torta *PSPieChart*.
- *PSSerie*: representa una serie en un gráfico. Define el(los) valor(es) de la serie, así como su nombre y varias características para su visualización como color, ancho, tipo de línea, etc. Definimos un tipo de serie para cada tipo de gráfico: *PSXYSerie* y *PSPieSerie*, ya que tienen comportamientos diferentes (una serie en un gráfico de torta es básicamente un valor mientras que una serie en un gráfico de curvas es una lista de valores).
- *PSAxis*: representa un eje en un gráfico.

4.3.2. *PSDrawer*

Estos objetos son los *dibujantes* de los objetos gráficos. Básicamente existe un tipo de dibujante para cada tipo de objeto que nos interesa dibujar. Por ejemplo, para dibujar los gráficos antes mencionados tenemos dos tipos de *ChartDrawer*: *XYPlotDrawer* y *PieChartDrawer*.

Para el caso de los modelos de percolación, o mejor dicho, sus grafos asociados, sólo definimos dibujantes para el caso de dos dimensiones dejando el de

tres para un trabajo a futuro. El más básico es el dibujante *SquareLatticeGraphDrawer*, siendo especializaciones de éste los dibujantes *BondPercolationGraphDrawer* y *SitePercolationGraphDrawer*.

Los dibujantes colaboran con algunas herramientas para realizar ciertas tareas.

4.3.3. *PSDrawerTool*

Para desacoplar algunas tareas específicas, como por ejemplo *pintar un cluster*, o *resaltar los vecinos de un vértice dado*, definimos algunos objetos que representan herramientas que pueden ser utilizadas por los dibujantes (*PSDrawerTool*). Esta separación es conveniente ya que las tareas llevadas a cabo por dichas herramientas involucran una lógica específica que excede a la responsabilidad de un dibujante (que es básicamente, dibujar ejes, vértices, curvas, etc.). Por ejemplo, la tarea de pintar un cluster a partir de un vértice dado, conlleva la exploración del grafo a partir de dicho vértice.

Este diseño nos permite definir nuevas herramientas y asociarlas a los dibujantes existentes, sin necesidad de modificarlos ya que las herramientas utilizan los servicios de dibujado definidos por los dibujantes.

4.4. Diseño de la interfaz gráfica de usuario (GUI)

Nuestro principal objetivo en el diseño de la interfaz gráfica de usuario (o *GUI, Grafic User Interface*) es lograr que esta muestre la información en forma clara y que sea cómoda de utilizar para el usuario. Estos objetivos son importantes ya que de ellos depende fuertemente la productividad de nuestro software.

Para el diseño y la construcción de la GUI utilizamos *Window Builder Pro 3.2*, en su versión para *Visual Smalltalk*. Esta herramienta genera código *Smalltalk* a partir de una especificación gráfica de los distintos paneles de una ventana y sus eventos asociados.

En la figura 18 vemos la pantalla principal de la GUI del simulador.

En la parte izquierda de la pantalla se especifica el modelo a simular proporcionando el tipo de modelo (*bond* o *site*), la política de apertura, la(s) probabilidad(es) de apertura, las dimensiones de la grilla (dadas por el tipo, *Square Lattice* o *Cubic Lattice*), el tamaño de la grilla y el patrón del grafo.

Por ejemplo, en la figura 19 podemos ver la especificación de un modelo de tipo *bond*, con una política isotrópica, sobre una grilla de tipo *Square Lattice* de tamaño 100×100 con el patrón *Square*.

4.4.1. Patrones

En la lista *Pattern* encontraremos los patrones conocidos básicos, como *square*, *bow-tie*, *hexagonal*, etc., así como los patrones específicos definidos en esta tesis, como $Square(V : \{1, k\}, H : \{1, k\})$, $Square(V : \{1, k\}, H : \{1\})$, etc.

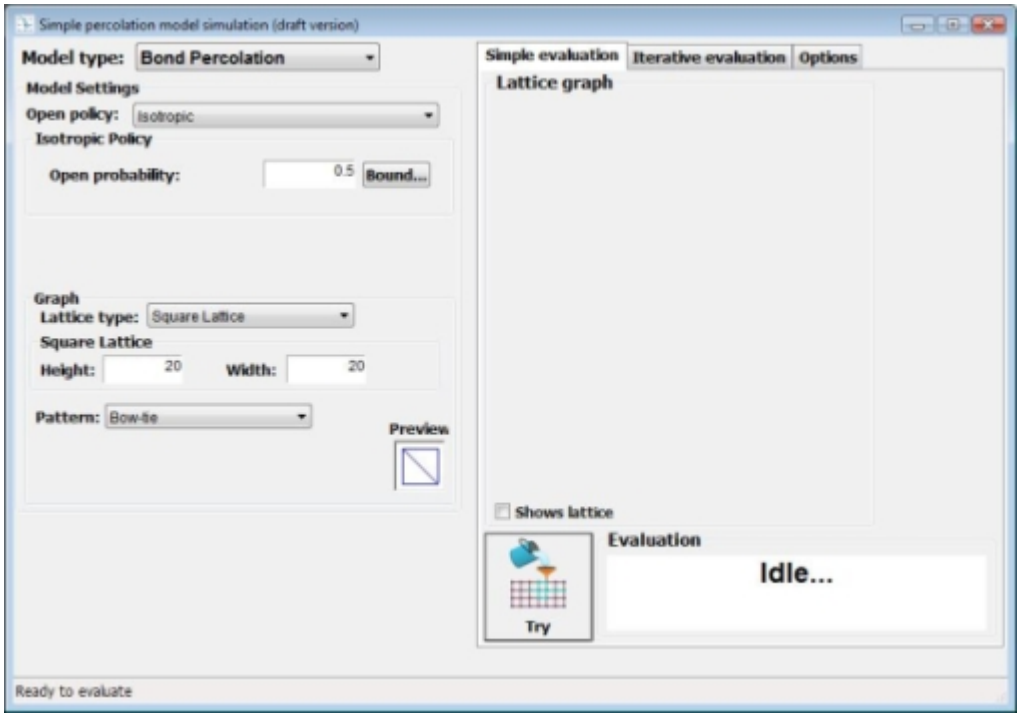

Figura 18: Pantalla principal.

Recordemos que existen dos tipos de patrones, unos especificados como subgrafos y otros especificados como el vecindario regular de un vértice. Para los primeros, como vemos en la figura 19, contamos una vista previa. En el caso de los segundos, puede ocurrir que requieran de algún parámetro adicional, como por ejemplo la distancia k. En tal caso, al seleccionar uno de dichos patrones, aparecerán los campos requeridos debajo de la lista, como vemos en la figura 20.

Creación de nuevos patrones. Adicionalmente a los patrones listados en la lista *Pattern* que encontramos en la figura 19, podemos crear nuevos patrones eligiendo en dicha lista la opción *New...* (ver figura 21). Al hacerlo debemos ingresar un nombre y luego definir gráficamente el patrón de tipo subgrafo, especificando su tamaño y dibujando los ejes entre sus vértices, como se muestra en la figura 22.

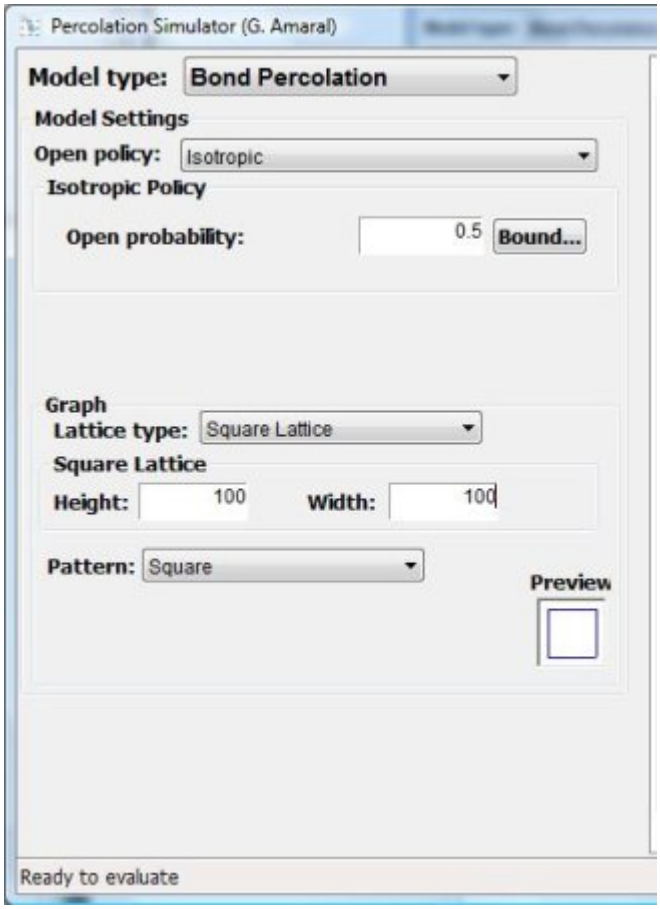

Figura 19: Especificación del modelo (parte izquierda de la pantalla principal).

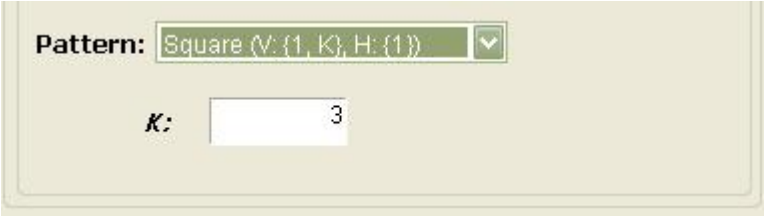

Figura 20: Especificación de un patrón con el parámetro *k*.

4.4.2. Critical Range Finder

A la derecha del campo *Open Probability* encontramos el botón *Bound*, mediante el cual podemos acceder la interfaz *Critical Range Finder* (ver figura 23)

Figura 21: Opción para crear un nuevo patrón.

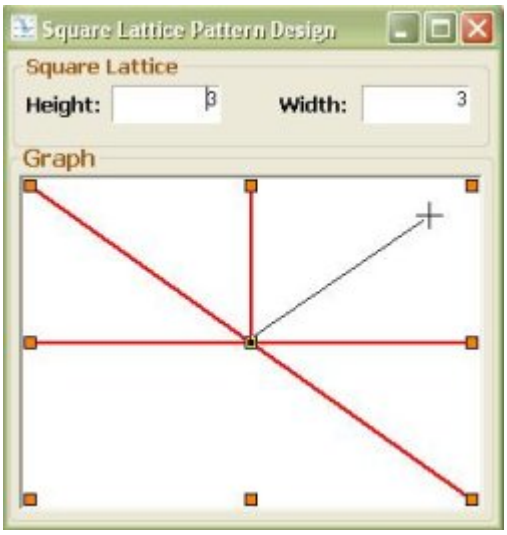

Figura 22: Creación gráfica de un nuevo patrón.

donde podemos ejecutar el algoritmo explicado en 3.7.

Para ello, luego de especificar los distintos parámetros, utilizamos el botón *Find!*, el cual inicia la búsqueda. El resultado se muestra en el gráfico de la parte inferior en forma de rango sobre el segmento de eje cartesiano comprendido entre 0 y 1.

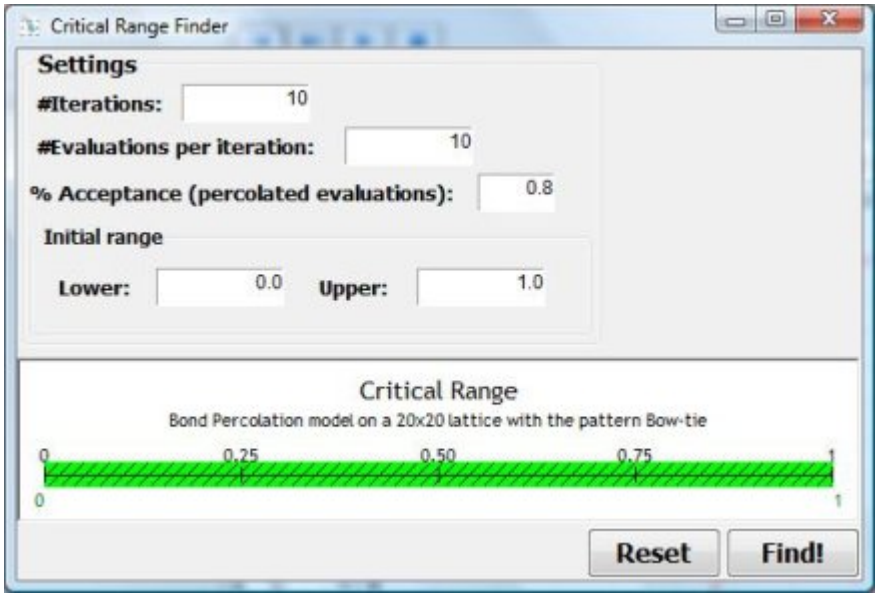

Figura 23: Pantalla del *Critical Range Finder*.

Por ejemplo, en la figura 24 podemos ver el resultado de realizar una búsqueda del rango crítico para un modelo bond *bow-tie* sobre una grilla de 20×20. La búsqueda fue configurada para 50 iteraciones, 10 evaluaciones por iteración y un porcentaje de aceptación del 80 %.

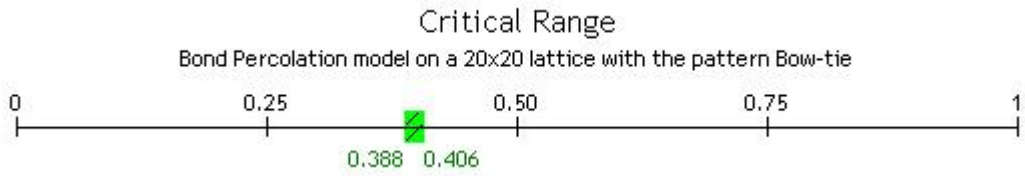

Figura 24: Resultado de una búsqueda utilizando *Critical Range Finder*.

En la parte derecha podemos visualizar diferentes pestañas para las simulaciones simples (*Simple simulation*), las simulaciones iterativas (*Iterative simulation*) y para opciones generales del modelo (*Options*).

4.4.3. Simple simulation

Permite realizar una simulación simple del modelo y visualizar el resultado de la misma. La simulación se inicia al hacer clic en el botón *Try*. Al finalizar la simulación, podremos ver el grafo resultante de la simulación, es decir, el grafo del modelo actualizado con los ejes (o vértices) abiertos y cerrados, mientras que en la parte inferior veremos un mensaje indicando si el resultado de la simulación fue positivo (existe un cluster percolante) o negativo (no existe tal cluster). Para una más rápida apreciación de este mensaje, el mismo se muestra en diferentes colores de acuerdo al resultado, verde para un resultado positivo y rojo para un resultado negativo.

En la figura 25 vemos el resultado de una simulación de un modelo con el patrón bow-tie.

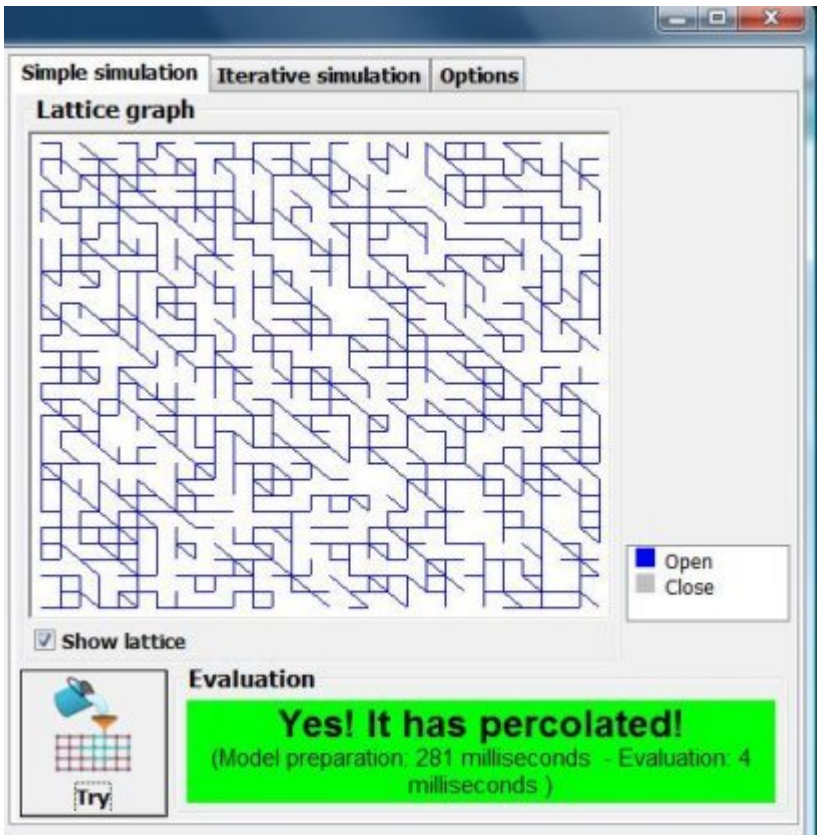

Figura 25: Simulación simple (parte derecha de la pantalla principal).

Adicionalmente, podemos cambiar varios aspectos del grafo utilizando el menú contextual. Por ejemplo, ver/ocultar los vértices, o ver/ocultar los ejes cerrados.

También podemos activar/desactivar algunas herramientas sobre el mismo grafo. Una de ellas, de gran utilidad, es la opción *Paint clusters (on selection)* que permite pintar un cluster a partir de un vértice como podemos ver en la figura 26.

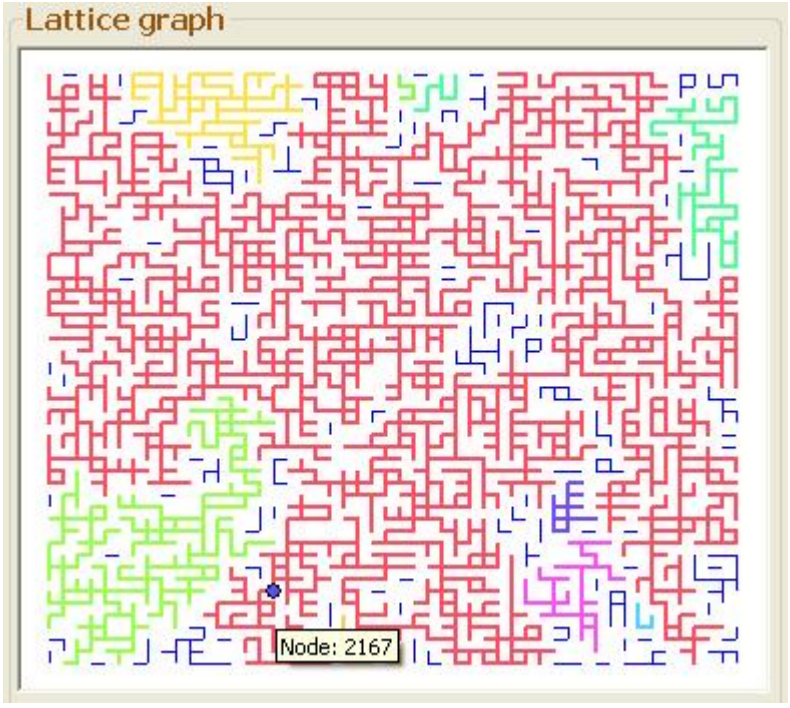

Figura 26: Algunos clusters pintados utilizando la opción *Paint clusters (on selection)* en un modelo *square* de 50×50.

Con la opción *Highlight neighbors* podemos activar otra herramienta que nos permite visualizar cúales son los vecinos de un vértice dado (ver figura 27). Esta herramienta es muy útil en el caso de los patrones en los cuales un vértice tiene más de un vecino en el sentido vertical (u horizontal) ya que los ejes que los unen son dibujados superpuestos y dificultan su identificación.

En el caso de querer evitar la visualización del grafo (aconsejable para grafos de tamaños mayores a 250×250), podemos utilizar la opción *Show lattice* en la parte inferior del dibujo del grafo.

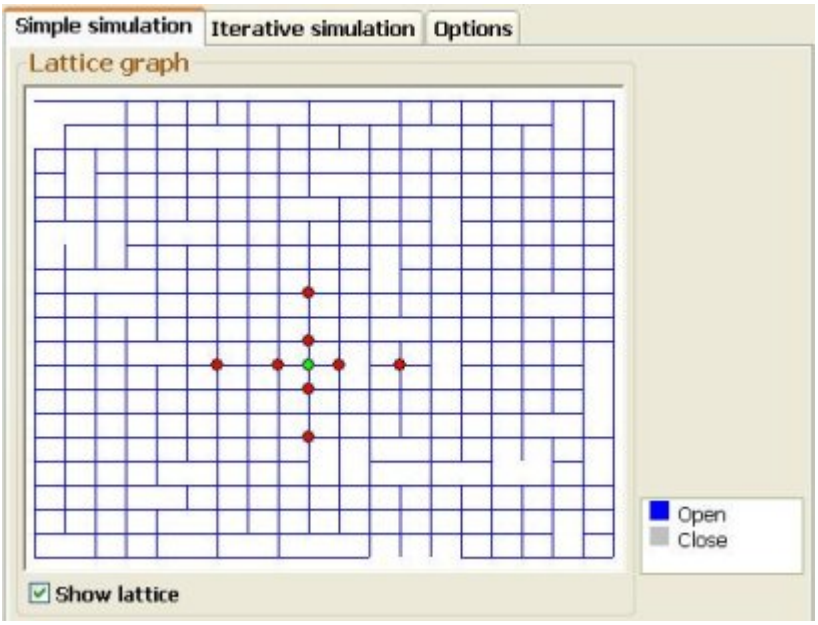

Figura 27: Vecinos de un vértice en un modelo *Square*($V : \{1, k\}, H : \{1, k\}$) activando la opción *Highlight neighbors*.

4.4.4. Iterative simulation

En esta página podemos realizar varios tipos de simulaciones iterativas y visualizar sus resultados en forma de gráficos de curvas.

Por ejemplo, podemos realizar una serie de simulaciones (sin ningun criterio) y ver los resultados sucesivos seleccionado el tipo *Simple Sampler* e ingresando el número de simulaciones que deseamos realizar.

Al igual que todos los tipos, las simulaciones se controlan con los botones de la figura 28.

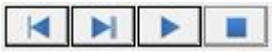

Figura 28: Botones de control de simulaciones.

De izquierda a derecha tenemos.

- **Retroceder:** retrocede la serie de simulaciones borrando los últimos resultados.
- **Avanzar un paso:** realiza la siguiente simulación. Si la marca *Refresh with every step* está seleccionada el gráfico es actualizado. Esta opción es muy útil para un seguimiento detallado de la evolución de las simulaciones, sin necesidad de esperar hasta que finalice la serie completa.

- **Avanzar:** realiza todas las simulaciones restantes y actualiza el gráfico. Actualiza el gráfico en cada paso si la marca *Refresh with every step* está seleccionada. Una vez que comienza la serie de simulaciones este botón puede utilizarse para pausar la serie.
- **Parar:** detiene las simulaciones y actualiza el gráfico con los datos obtenidos hasta el momento.

En la figura 29 vemos el resultado de realizar 20 simulaciones de un modelo de tipo bond sobre una grilla de 20×20 con el patrón bow-tie y con una probabilidad de apertura de 0,5. En la parte inferior podemos ver el gráfico donde la curva seleccionada es el estimador de la función $\theta(p)$ (*Percolating Cluster Probability*).

Notemos que una de las variables que podemos consultar es el tiempo de cada simulación en milisegundos (*Evaluation Time (ms)*). Esto es muy útil ya que permite analizar el comportamiento (en tiempo) de los diferentes algoritmos sobre un modelo particular.

Podemos cambiar varios aspectos del gráfico de curvas mediante el menú contextual. Por ejemplo, propiedades de las curvas como color, estilo, descripción, etc., así como también las escalas de los ejes. También podemos agregar curvas de tendencia y promedio para alguna de las curvas del gráfico, o marcas en alguno de los ejes. Adicionalmente podemos agregar el gráfico a los *User charts* o combinarlo con otro gráfico agregado previamente.

Otro tipo de simulaciones iterativas es el *Variable Walker*. Para realizar este tipo de simulaciones debemos especificar la variable de entrada que queremos variar y el rango e incremento de los valores de la variable. Adicionalmente, podemos especificar si deseamos realizar un promedio para cada valor con la marca *Calculate average* y sobre cuántas simulaciones de muestra (*Average samples*).

En la figura 30 vemos el resultado de realizar una serie de simulaciones para un modelo de tipo bond sobre una grilla de 20×20 con el patrón bow-tie. La curva esta formada por el promedio de 20 evaluaciones para cada valor de la variable *Open Probability* en el rango $[0, 1]$ con un incremento de 0,05.

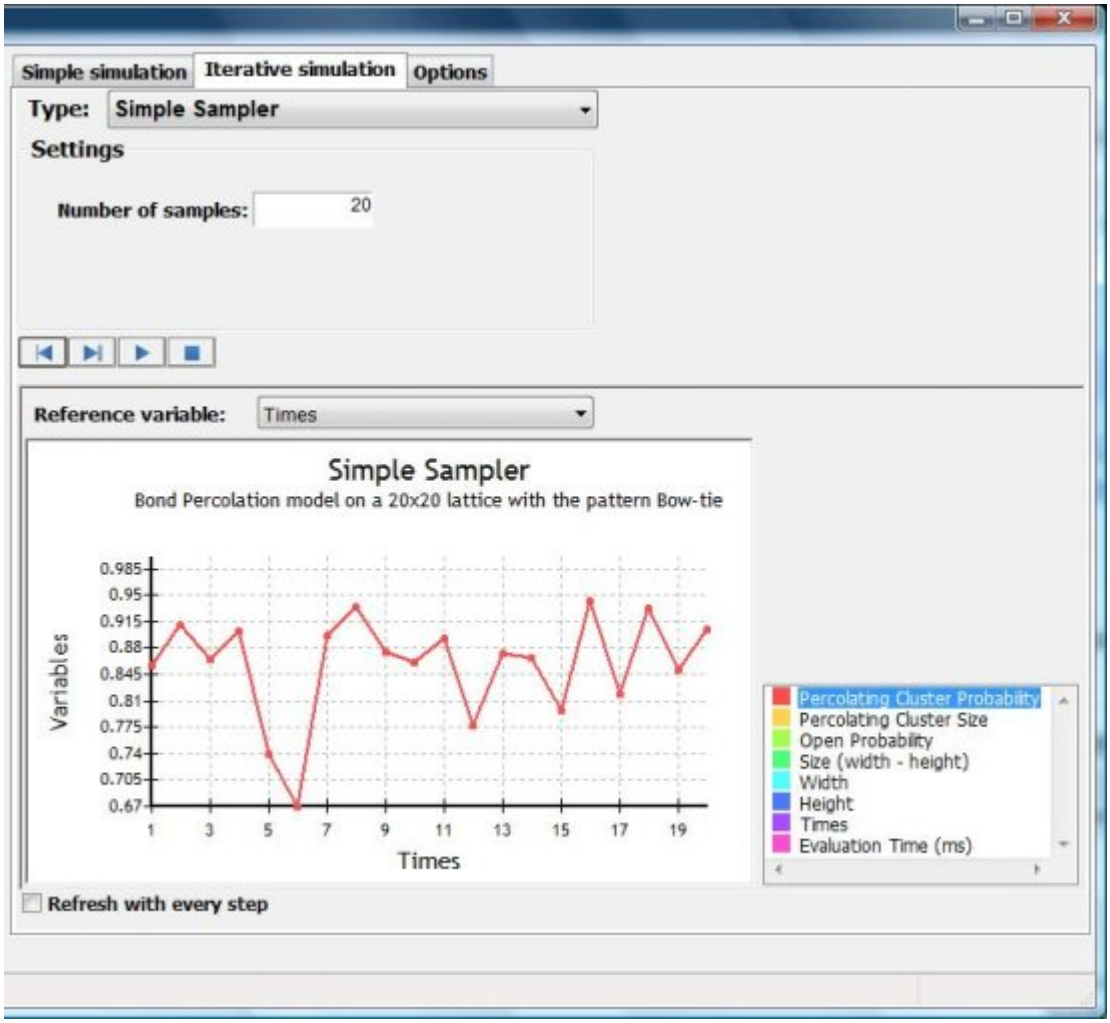

Figura 29: Simulación iterativa. *Simple Sampmler*.

5. Simulaciones

En esta sección describimos las simulaciones realizadas tanto para modelos de percolación con resultados teóricos conocidos, como para aquellos modelos estudiados en esta tesis, para los cuales aún no se conocen resultados teóricos.

Comenzamos con aquellos cuyo resultado teórico es conocido dado que son de gran utilidad para testear el funcionamiento de nuestro simulador. Sin embargo, como mencionamos anteriormente (ver 3.1), debemos tener en cuenta que los resultados teóricos estan referidos a modelos infinitos mientras que nuestras simulaciones trabajan sobre modelos finitos. Este factor puede oca-

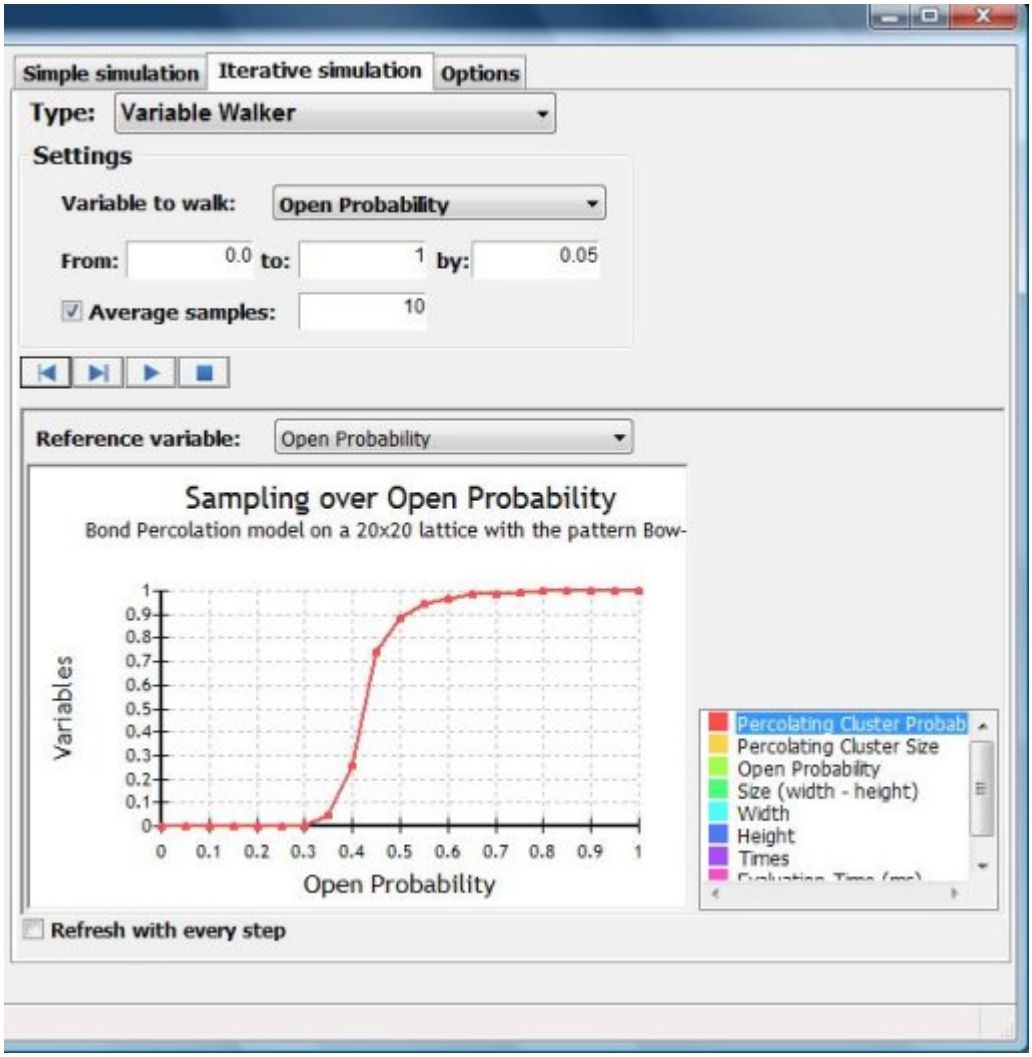

Figura 30: Simulación iterativa. *Variable Walker*.

sionar diferencias entre los resultados. No obstante, una diferencia significativa es un buen indicador de que nuestro simulador no funciona adecuadamente.

5.1. Modelos isotrópicos en $\mathbb{L}^2$

Estos modelos se caracterizan por tener un único parámetro (p) que determina la apertura o cierre de sus ejes (caso bond) o sus vértices (caso site).

A continuación resumimos los resultados de algunas simulaciones para varios modelos conocidos sobre diferentes tamaños de grilla. Para las corre-

spondientes a las regiones críticas utilizamos el algoritmo explicado en 3.7 para 100 (en algunos casos 50) iteraciones, 10 evaluaciones (en cada límite, inferior y superior) por iteración y un porcentaje de aceptación del 80 %. Adicionalmente realizamos un análisis estadístico del comportamiento del estimador de $\theta(p)$ realizando varias simulaciones para diferentes valores de p y tomando un promedio del estimador para cada valor de p.

Una aclaración importante es la siguiente. En todas las pruebas podemos ver que las regiones críticas resultantes no necesariamente encierran los valores conocidos de la probabilidad crítica. Consideramos que esto se debe a que estamos trabajando con un porcentaje de aceptación del 80 % y con solo 10 evaluaciones en cada valor. Es decir, si 8 de 10 evaluaciones cumplen que el modelo no percola (respectivamente percola), entonces el algoritmo avanzará el límite inferior (respectivamente superior) hacia la derecha (respectivamente izquierda). Una alternativa puede ser aumentar el porcentaje de aceptación. Esta alternativa es más que razonable pero para que la región resultante sea de un ancho similar al obtenido con un 80 % deberíamos realizar más evaluaciones por iteración para que tal aceptación tenga más sentido, aumentando así la cantidad de evaluaciones totales y en definitiva el tiempo de la prueba. Elegimos entonces una relación de compromiso entre precisión y tiempo, considerando que las pruebas iniciales arrojaron valores muy buenos para la configuración mencionada de 100 iteraciones, 10 evaluaciones por iteración y un porcentaje de aceptación del 80 %.

5.1.1. Square

Probablemente el modelo más estudiado es el modelo *square* de tipo bond (ver figura 31). En 1955 se conjeturó que la probabilidad crítica de este modelo era $\frac{1}{2}$. En 1960 T. Harris demostro que $\theta(\frac{1}{2}) = 0$ ([HAR/60]), derivándose de este hecho que $p_c \geq \frac{1}{2}$. Luego de algunos intentos y de un período de estancamiento en la teoría de la percolación, recién a fines de los '70s, Harry Kesten logró demostrar que $p_c = \frac{1}{2}$ ([KES/80]).

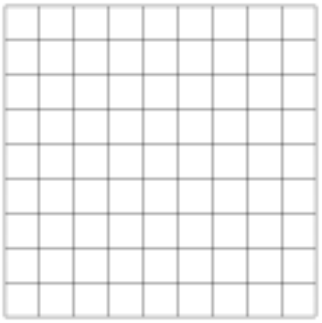

Figura 31: Estructura del modelo *square*.

En el cuadro 3 vemos las regiones críticas resultantes de la simulación para

el caso bond.

Tamaño	Límite Inferior	Límite Superior
10×10	0,46667849	0,51512855
20×20	0,46448595	0,51713766
30×30	0,48169255	0,51323220
40×40	0,49378969	0,51980976
50×50	0,48458348	0,51101252
100×100	0,49238418	0,50627266
150×150	0,49785232	0,49999999
200×200	0,50000687	0,50273132
250×250	0,49851074	0,50332031
300×300	0,49693299	0,50384227
350×350	0,49739208	0,50136560
400×400	0,49853615	0,50223577
450×450	0,49809944	0,50229566
500×500	0,49712696	0,50205078

Cuadro 3: Regiones críticas del modelo bond *square*.

Como se puede apreciar las regiones críticas resultantes de las diferentes simulaciones estan centradas (en promedio) alrededor de $\frac{1}{2}$. Es notable como a medida que aumenta el tamaño de la grilla las regiones son más pequeñas. Este hecho muestra que cuanto mayor es el tamaño de la grilla, mejores son las estimaciones en base a resultados estadísticos.

En la figura 32 vemos un gráfico con los resultados promedio del estimador de $\theta(p)$ para p variando de 0 a 1 con un incremento de 0,05. Para cada valor de p se tomó un promedio de 20 muestras. El modelo considerado es de 100×100. El hecho de que la curva se haga positiva alrededor de 0,5 concuerda con el resultado teórico.

En lo que respecta al caso site, existen diversas fuentes que hacen mención de la probabilidad crítica. Si bien existen ligeras diferencias entre tales fuentes, todas coinciden en que la probabilidad crítica es aproximadamente 0,59274 (ver por ejemplo [FDB/08] y [NEZ/00]).

En el cuadro 4 podemos ver las regiones críticas para el caso site.

Al igual que en el caso bond, se nota una clara reducción de las regiones a medida que aumentamos de tamaño de grilla. Del mismo modo, en la figura 33 vemos un gráfico con los resultados promedio (de 20 muestras) del estimador de $\theta(p)$ para p variando de 0 a 1 con un incremento de 0,05. Podemos apreciar que la curva se hace positiva llegando a 0,6.

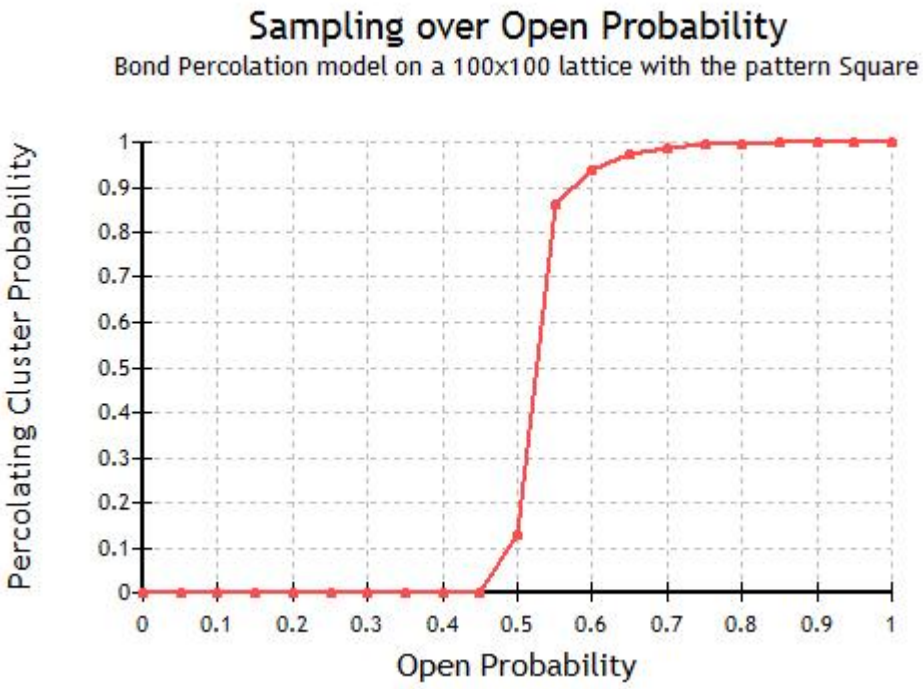

Figura 32: Estimador de $\theta(p)$ para un modelo bond *square* de 100×100.

Tamaño	Límite Inferior	Límite Superior
10×10	0,53750377	0,59452819
20×20	0,56651725	0,60700673
30×30	0,56719988	0,61424797
40×40	0,57503099	0,60439453
50×50	0,57560937	0,59997558
100×100	0,58437500	0,59687423
150×150	0,58659257	0,59686273
200×200	0,58847656	0,59755859
250×250	0,59063110	0,59531173
300×300	0,59072340	0,59490928
350×350	0,58935254	0,59296869
400×400	0,59375000	0,59587373
450×450	0,59096833	0,59372405
500×500	0,58952285	0,59570494

Cuadro 4: Regiones críticas del modelo site *square*.

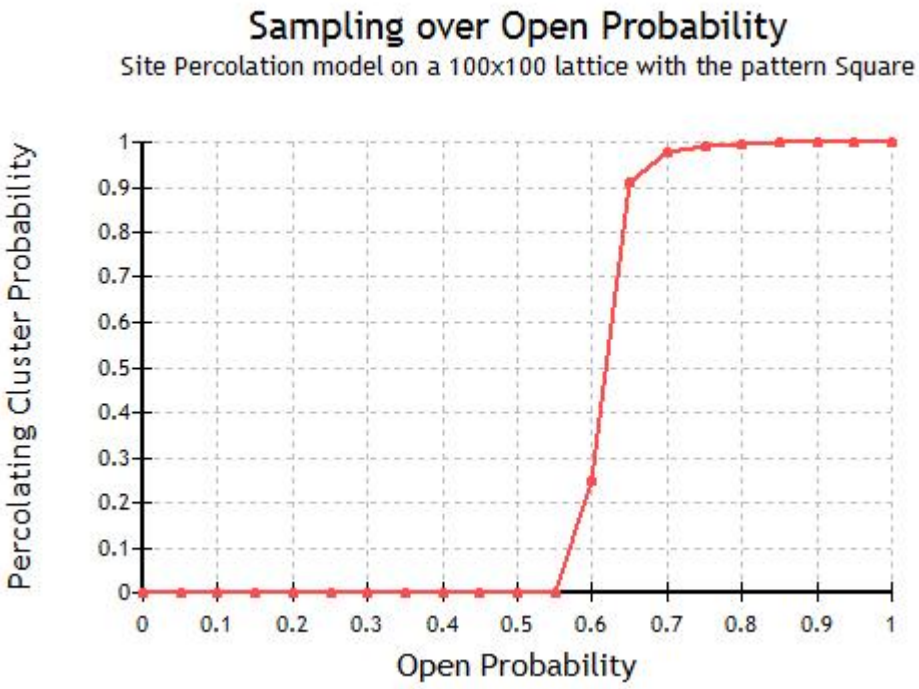

Figura 33: Estimador de $\theta(p)$ para un modelo site *square* de 100×100.

5.1.2. Bow-tie

Otro modelo famoso en dos dimensiones es el denominado *bow-tie*. Recordemos que su estructura es como la que se muestra en la figura 34.

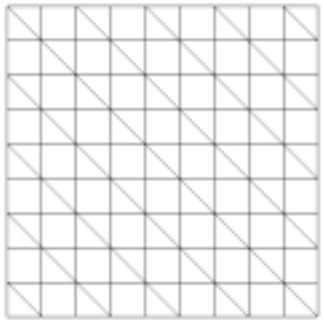

Figura 34: Estructura del modelo *bow-tie*.

La probabilidad crítica de este modelo en el caso bond es la raíz (única) en el intervalo $(0, 1)$ del polinomio $1 - p - 6p^2 + 6p^3 - p^5$. Dicho valor es aproxi-

madamente 0,404518 ([WIE/84]). En el caso site la probabilidad crítica es aproximadamente 0,5472 ([VDM/97]).

En los cuadros 5 y 6 vemos los resultados de las simulaciones para ambos casos, bond y site.

Tamaño	Límite Inferior	Límite Superior
10×10	0,37547002	0,42375488
20×20	0,38282697	0,42301541
30×30	0,38891706	0,41473939
40×40	0,39387817	0,41249903
50×50	0,39029379	0,41532150
100×100	0,40007172	0,41509543
150×150	0,40013027	0,40779775
200×200	0,40118153	0,40468750
250×250	0,40352936	0,40478287
300×300	0,40195580	0,40702818
350×350	0,40205785	0,40653939
400×400	0,40315611	0,40604847
450×450	0,40217938	0,40527336
500×500	0,40317466	0,40591370

Cuadro 5: Regiones críticas del modelo bond *bow-tie*.

Tamaño	Límite Inferior	Límite Superior
10×10	0,46389262	0,57968597
20×20	0,50129738	0,57460586
30×30	0,52697753	0,56249847
40×40	0,51958012	0,56789398
50×50	0,52780305	0,56867314
100×100	0,53618024	0,55935834
150×150	0,53720956	0,55685142
200×200	0,54377746	0,54999539
250×250	0,54229473	0,55272705
300×300	0,54375028	0,55155639
350×350	0,54440956	0,54999993
400×400	0,54412995	0,55065798
450×450	0,54435703	0,54958337
500×500	0,54473368	0,54921874

Cuadro 6: Regiones críticas del modelo site *bow-tie*.

En las figuras 35 y 36 podemos apreciar los resultados del estimador de $\theta(p)$ sobre p, variando de p de 0 a 1 con un incremento de 0,05 (tomando un

promedio de 20 muestras en cada punto). En ambos casos vemos que el primer valor positivo se aproxima a los valores críticos conocidos (0,404518 para el caso bond y 0,5472 para el caso site).

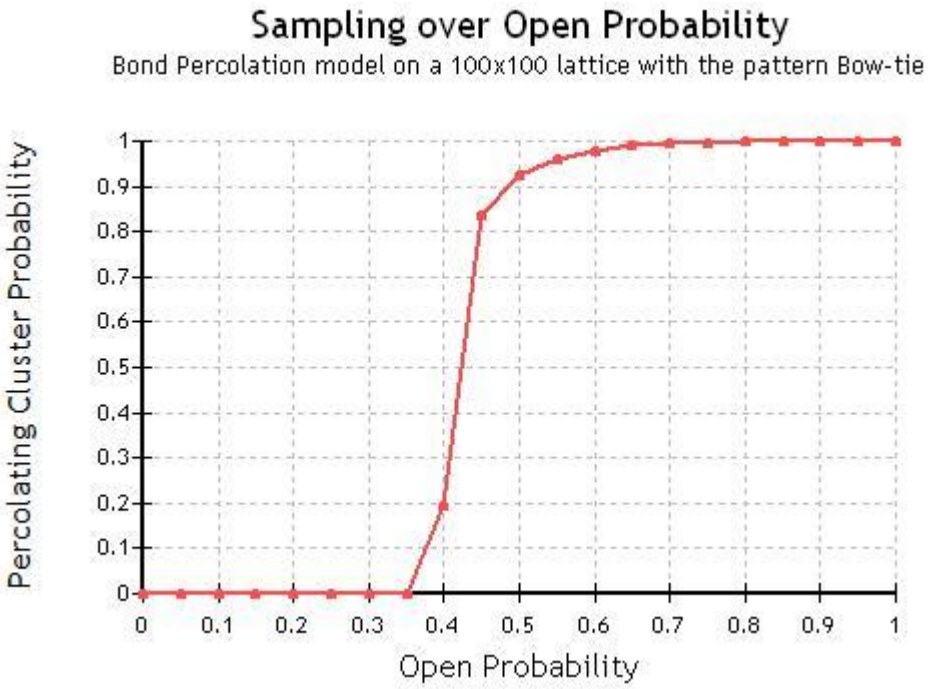

Figura 35: Estimador de $\theta(p)$ para un modelo bond *bow-tie* de 100×100.

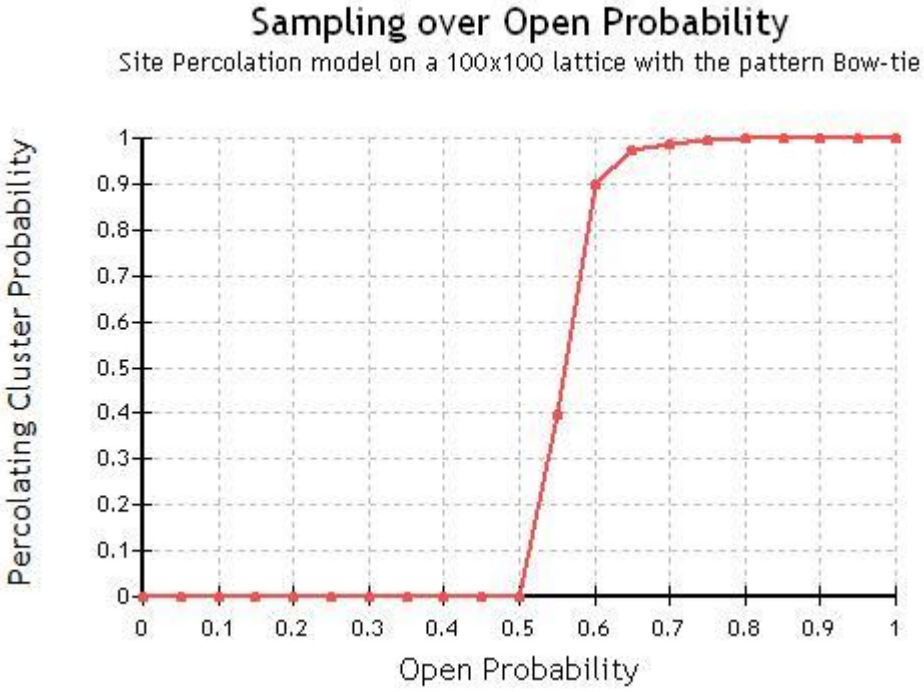

Figura 36: Estimador de $\theta(p)$ para un modelo site *bow-tie* de 100×100.

5.1.3. Hexagonal

El modelo denominado *Hexagonal* se caracteriza por tener la estructura de la figura 37.

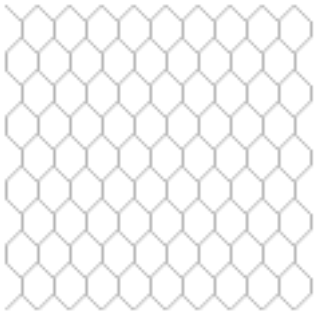

Figura 37: Estructura del modelo *hexagonal*.

La probabilidad crítica de este modelo en el caso bond es $1 - 2sin(\frac{\pi}{18})$ ([SKE/98], [GRI/99]) (aproximadamente 0,6527), mientras que en el caso site

la probabilidad crítica es aproximadamente 0,6970 ([ZIS/99]).

En los cuadros 7 y 8 vemos los resultados de las simulaciones para los casos bond y site.

Tamaño	Límite Inferior	Límite Superior
10×10	0,58496260	0,69661427
20×20	0,60630345	0,67357625
30×30	0,63190155	0,66197256
40×40	0,62442541	0,66480963
50×50	0,63360758	0,65541800
100×100	0,64376279	0,65621289
150×150	0,65058670	0,66125643
200×200	0,64707031	0,64999961
250×250	0,64688116	0,65544414
300×300	0,65000615	0,65600460
350×350	0,65001227	0,65495241
400×400	0,65009925	0,65312333
450×450	0,65009768	0,65497651
500×500	0,64922345	0,65538815

Cuadro 7: Regiones críticas del modelo bond *hexagonal*.

Tamaño	Límite Inferior	Límite Superior
10×10	0,65625153	0,72498468
20×20	0,65344020	0,69609184
30×30	0,66964116	0,69999682
40×40	0,67232747	0,70933512
50×50	0,66944769	0,71815503
100×100	0,68643842	0,69838561
150×150	0,68829040	0,69989624
200×200	0,68766059	0,69920952
250×250	0,68653116	0,69986482
300×300	0,68957734	0,69958021
350×350	0,69385148	0,70038810
400×400	0,69105520	0,69989566
450×450	0,69226114	0,69667966
500×500	0,69136638	0,69687499

Cuadro 8: Regiones críticas del modelo site *hexagonal*.

En las figuras 38 y 39 vemos los resultados del estimador de $\theta(p)$ sobre p. En ambos casos podemos observar que el primer valor positivo es aproximadamente la probabilidad crítica conocida.

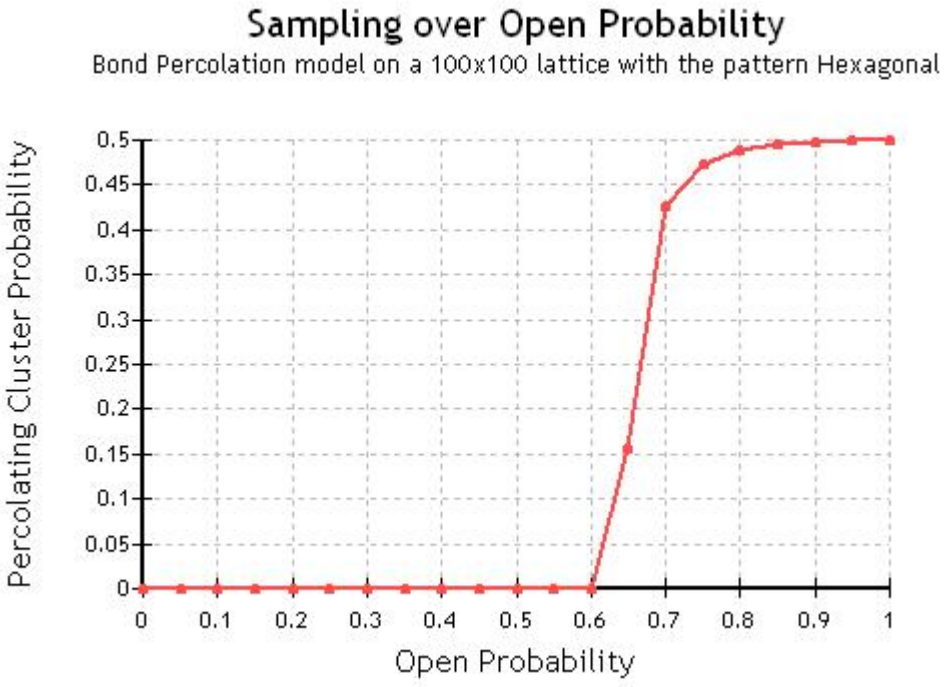

Figura 38: Estimador de $\theta(p)$ para un modelo bond *hexagonal* de 100×100.

5.1.4. Triangular

El modelo denominado *Triangular* tiene la estructura de la figura 40.

La probabilidad crítica de este modelo en el caso bond es $2sin(\frac{\pi}{18})$ ([SKE/98]) (aproximadamente 0,3472), mientras que en el caso site la probabilidad crítica es 0,5 ([GRI/99]).

En los cuadros 9 y 10 vemos los resultados de las simulaciones para los casos bond y site.

En las figuras 41 y 42 vemos el estimador de $\theta(p)$ sobre p donde el primer valor positivo coincide aproximadamente con las probabilidad críticas conocidas.

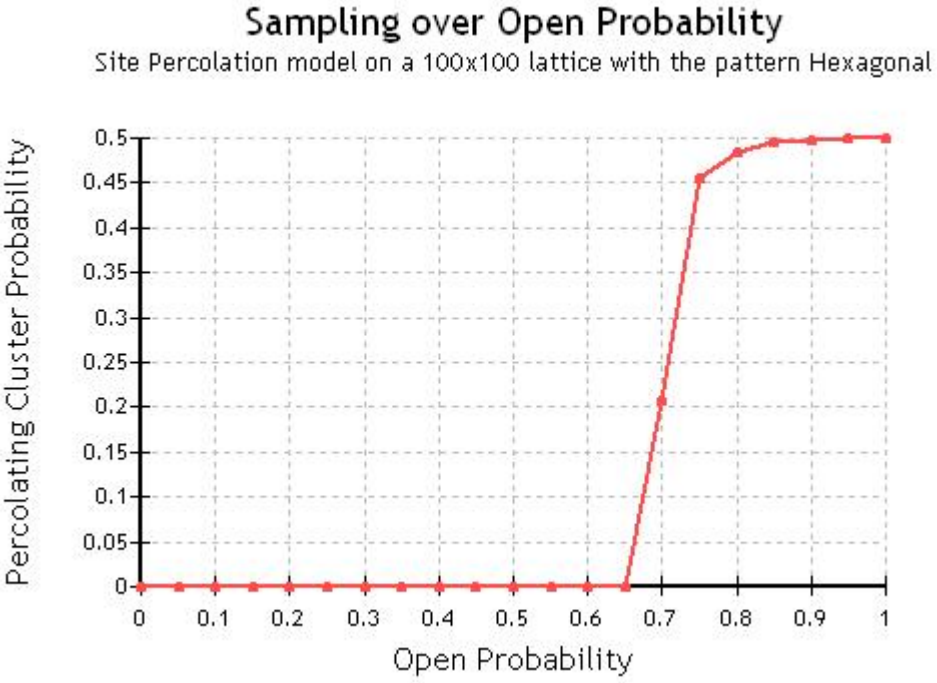

Figura 39: Estimador de $\theta(p)$ para un modelo site *hexagonal* de 100×100.

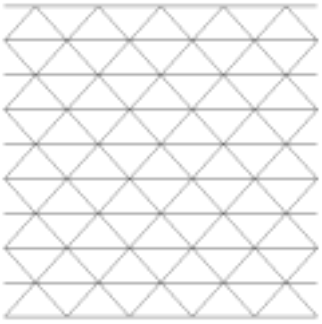

Figura 40: Estructura del modelo *Triangular*.

5.2. Modelos anisotrópicos en $\mathbb{L}^2$

Estos modelos se caracterizan por tener más de un parámetro para determinar la apertura o cierre de los ejes de acuerdo a la dirección de los mismos (vertical, horizontal, diagonal). Notemos que no tiene sentido hablar de anisotropía en los modelos de tipo site ya que los vértices no tienen dirección.

Dado que estos modelos son mucho más generales que los modelos isotrópi-

Tamaño	Límite Inferior	Límite Superior
10×10	0,27622843	0,37499275
20×20	0,31699301	0,37434482
30×30	0,32131362	0,36789962
40×40	0,32658882	0,36044573
50×50	0,33281817	0,36109478
100×100	0,33812338	0,35696475
150×150	0,34218754	0,35145261
200×200	0,34024583	0,34980235
250×250	0,34377460	0,34992057
300×300	0,34376565	0,34838714
350×350	0,34375009	0,34996807
400×400	0,34384849	0,34916991
450×450	0,34376907	0,34990202
500×500	0,34367947	0,34843630

Cuadro 9: Regiones críticas del modelo bond *triangular*.

Tamaño	Límite Inferior	Límite Superior
10×10	0,41338015	0,54370093
20×20	0,43779322	0,52490191
30×30	0,45746241	0,53181025
40×40	0,47500000	0,52108730
50×50	0,46920192	0,50781247
100×100	0,48168383	0,49999999
150×150	0,48753862	0,50817751
200×200	0,48766191	0,50644334
250×250	0,49103516	0,50588889
300×300	0,49272617	0,49999990
350×350	0,49069939	0,50291668
400×400	0,49541018	0,50404598
450×450	0,49531577	0,50155996
500×500	0,49536221	0,50151367

Cuadro 10: Regiones críticas del modelo site *triangular*.

cos, y por lo tanto existe una infinidad de variantes, no contamos con resultados teóricos ni estudios numéricos sobre los mismos. Por otra parte, estos modelos son quizas los que más aplicaciones prácticas tengan dado que su especificidad por lo general refleja características del problema o situación real modelada.

Es importante notar que aquí no podemos hablar de una *única* probabilidad crítica, ya que tendremos (potencialmente) muchas probabilidades críticas. Mas concretamente, tendremos una probabilidad crítica para cada valor puntual del

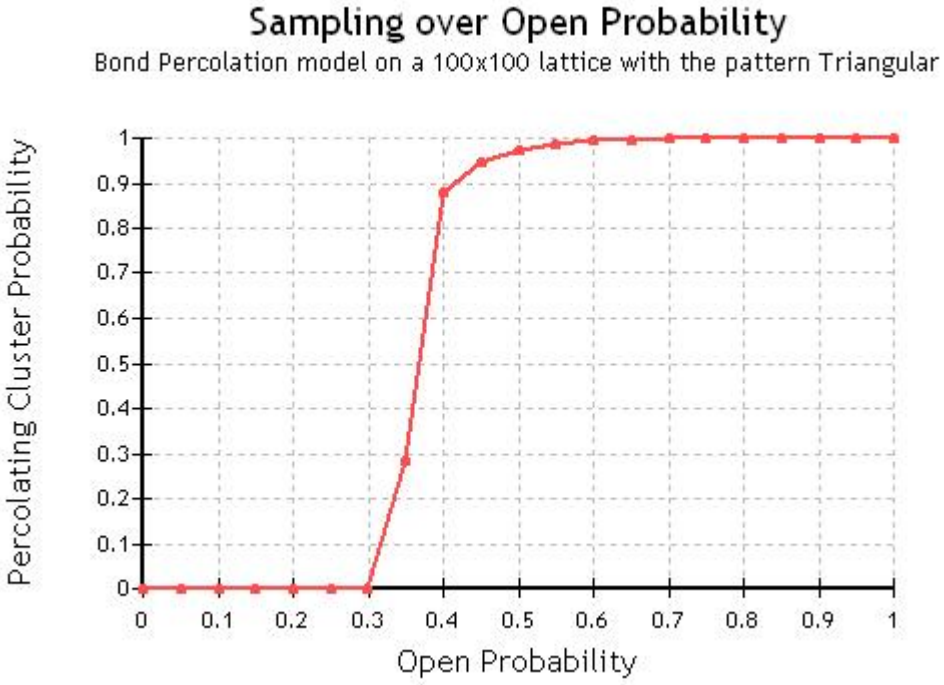

Figura 41: Estimador de $\theta(p)$ para un modelo bond *triangular* de 100×100.

resto de las probabilidades. Por ejemplo, si tenemos dos parámetros de apertura p_H y p_V en un modelo de tipo bond, con p_H para los ejes horizontales y p_V para los ejes verticales, entonces para cada valor particular de p_H tendremos una probabilidad crítica p_{c_V} para el parámetro p_V.

A continuación proponemos algunos modelos anisotrópicos y mostramos los resultados de las simulaciones de los mismos. Estos modelos no tienen una motivación en una situación real si no que resultan del proceso de testeo del simulador implementado.

5.2.1. Square Vertical/Horizontal

Partiendo del modelo más simple y estudiado, bond *square*, elegimos una política de apertura anisotrópica de dos parámetros: p_H para los ejes horizontales y p_V para los verticales. En las figuras 43, 44, 45 y 46 podemos ver el diferentes combinaciones de ambas probabilidades.

Si bien las combinaciones pueden ser infinitas, existe un interrogante interesante de responder: si dejamos fija una de ambas probabilidades, ¿cuál es la probabilidad crítica para la otra?. Por ejemplo, supongamos que dejamos fija $p_H = 0{,}25$, ¿cuál será la probabilidad crítica de p_V?. Para responder a esto podemos utilizar el mismo algoritmo que utilizamos para los modelos isotrópicos.

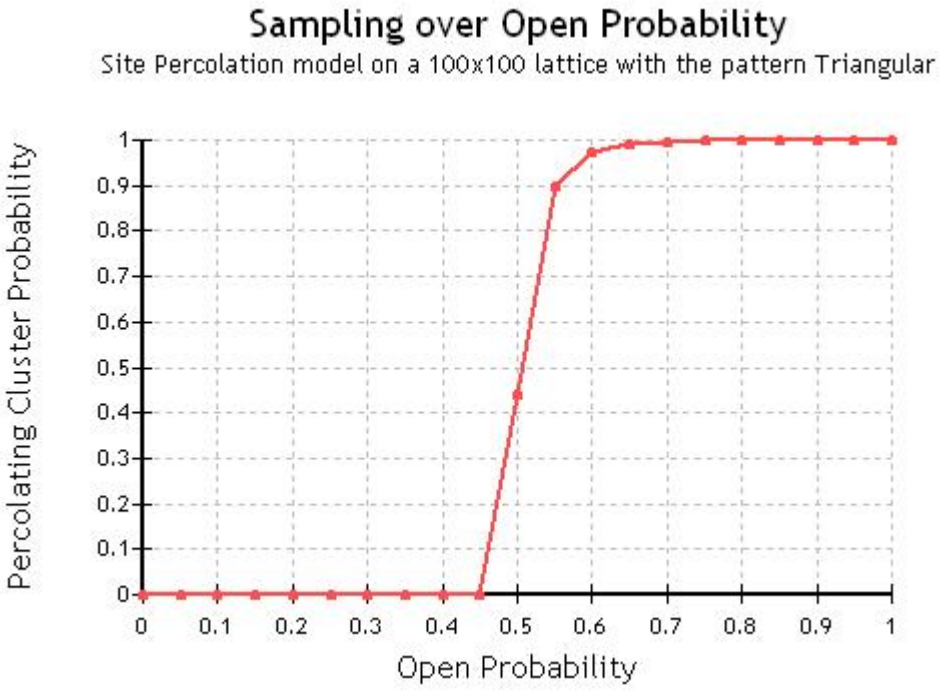

Figura 42: Estimador de $\theta(p)$ para un modelo site *triangular* de 100×100.

En la figura 47 podemos observar el resultado de la ejecución de tal algoritmo sobre un modelo *square* de 100×100, donde vemos que tal probabilidad se ubica en las cercanías de 0,72.

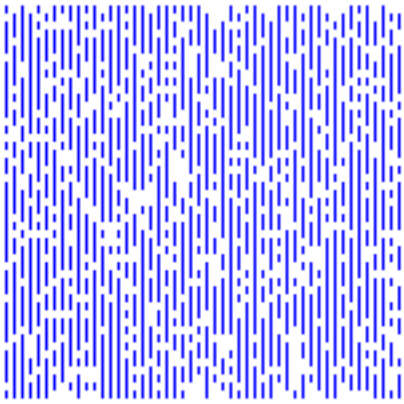

Figura 43: $p_H = 0$ y $p_V = 0{,}75$.

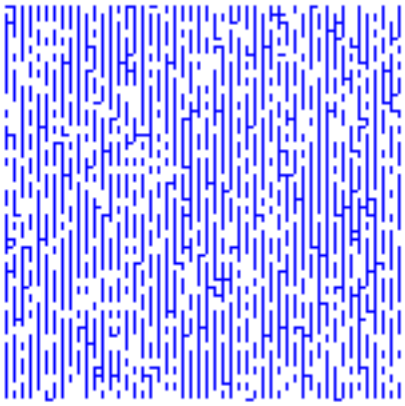

Figura 44: $p_H = 0{,}05$ y $p_V = 0{,}75$.

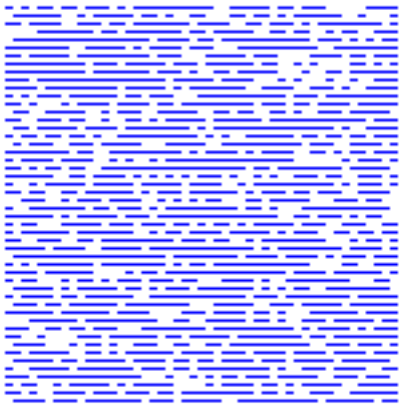

Figura 45: $p_H = 0{,}75$ y $p_V = 0$.

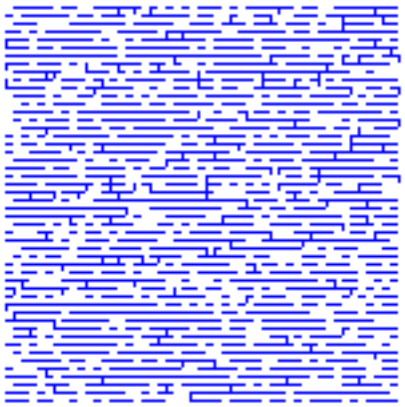

Figura 46: $p_H = 0{,}75$ y $p_V = 0{,}05$.

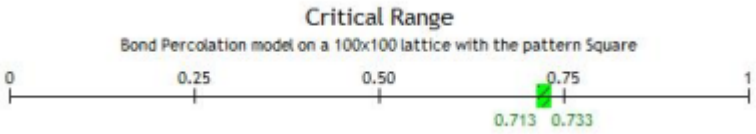

Figura 47: Región crítica para un modelo bond *square* con $p_H = 0{,}25$ fijo.

Si realizamos el mismo análisis dejando $p_V = 0{,}25$ fija y buscando caminos de percolación en el sentido horizontal obtenemos un resultado similar para la probabilidad crítica de p_H (figura 48). Este resultado es bastante intuitivo ya que este experimento coincide con el anterior con la diferencia de que cambiamos la orientación.

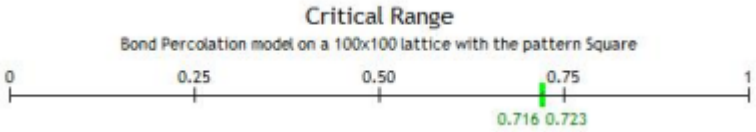

Figura 48: Región crítica para un modelo bond *square* con $p_V = 0{,}25$ fijo.

5.2.2. Análisis de alcance

Otro análisis que podemos realizar sobre un modelo con las características mencionadas es el siguiente. Supongamos que en lugar de buscar un cluster percolante nos interesara estudiar el conjunto de vértices alcanzados partiendo desde un vértice dado. Por ejemplo, supongamos que iniciamos la búsqueda en el vértice 1. Tomamos un vértice cualquiera $x = (x_1, x_2)$ con $x_1 < x_2$, y consideramos el vértice *espejo* $x' = (x_2, x_1)$[21]. Sean $P(1 \leftrightarrow x)$ y $P(1 \leftrightarrow x')$ las probabilidades de que exista un camino abierto entre el vértice 1 y los vértices x y x' respectivamente. Resulta interesante estudiar cómo se comportan estas probabilidades respecto de los parámetros p_H y p_V. Un artículo sobre este tema es [LIM/04].

Intuitivamente podemos pensar que si $p_H > p_V$ entonces $P(1 \leftrightarrow x) > P(1 \leftrightarrow x')$ (de forma análoga si $p_H < p_V$ entonces $P(1 \leftrightarrow x) < P(1 \leftrightarrow x')$). Sin embargo, este hecho no está demostrado más que para algunos casos particulares.

Para reforzar esta conjetura podemos realizar un estudio estadístico de los resultados de las simulaciones. En este caso, una simulación consitirá en expandir el cluster desde el vértice 1 y contabilizar los vértices alcanzados. Al cabo de varias simulaciones tendremos el número de veces que cada vértice

[21] En la representación de la grilla elegida, el vértice 1 es aquel ubicado en la esquina superior izquierda de la grilla y sus coordenadas son $(1, 1)$. Por otra parte, las coordenadas de un vértice son tomadas con respecto a dicho origen, es decir, aumentan hacia la derecha y hacia abajo. Ver la sección 3.4 para más detalles.

fue alcanzado. Con estos resultados podemos realizar un mapa coloreando los vértices de forma conveniente para poder extraer alguna conclusión.

Una forma de colorear los vértices de acuerdo a un número asociado a cada uno de ellos es mediante una gama de colores. En nuestro caso, si utilizamos una gama de 5 colores podemos particionar el número de simulaciones en 5 rangos, asignando un color a cada rango. De esta manera podemos analizar el mapa observando los colores de x y el de x'.

Otra alternativa es colorear un vértice en función del número de veces que fue alcanzado respecto del número de veces que fue alcanzado su *espejo*. Por ejemplo, si x fue alcanzado N_x veces y x' lo fue $N_{x'}$, podemos colorearlos utilizando 3 colores diferentes dependiendo de si $N_x = N_{x'}$, $N_x < N_{x'}$ ó $N_x > N_{x'}$. Adicionalmente, podemos *relajar* la condición $N_x = N_{x'}$, convirtiéndola en $N_x \approx N_{x'}$, donde $N_x \approx N_{x'}$ si $\frac{|N_x - N_{x'}|}{N} < \delta$ para algún porcentaje δ pequeño.

En la figura 49 podemos ver un ejemplo de este método para el resultado de 1000 simulaciones con $p_H = 0{,}4$ y $p_V = 0{,}6$. Los colores utilizados para colorear un vértice x son: **verde** si $N_x > N_{x'}$, **amarillo** si $N_x \approx N_{x'}$ (con $N_x \approx N_{x'}$ si $\frac{|N_x - N_{x'}|}{N} < 0{,}001$) y **rojo** si $N_x < N_{x'}$. En él podemos observar que los vértices debajo de la diagonal han sido alcanzados (en su mayoría) un mayor número de veces que los que estan por encima de la diagonal.

Realizando este mismo experimento sobre un modelo de 100×100 obtenemos un resultado similar, como podemos ver en la figura 50.

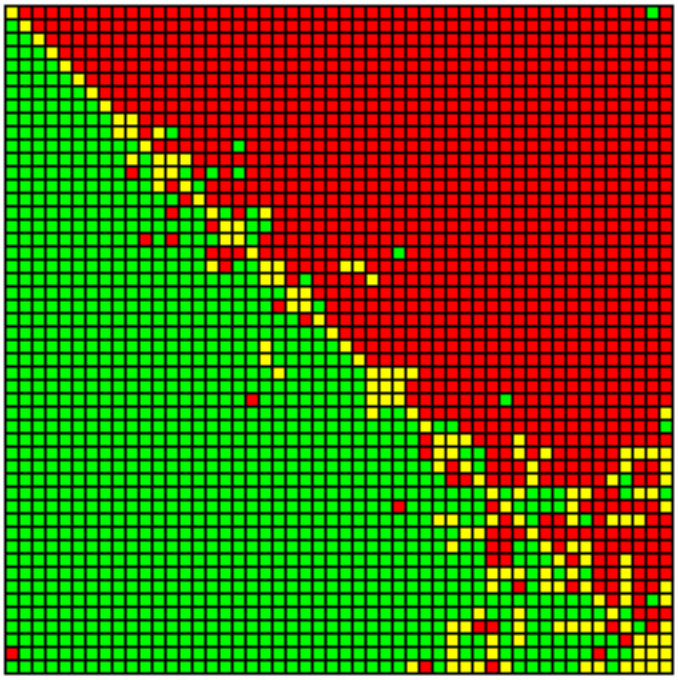

Figura 49: Resultado de 1000 simulaciones de un modelo de 50×50 con $p_H = 0{,}4$ y $p_V = 0{,}6$.

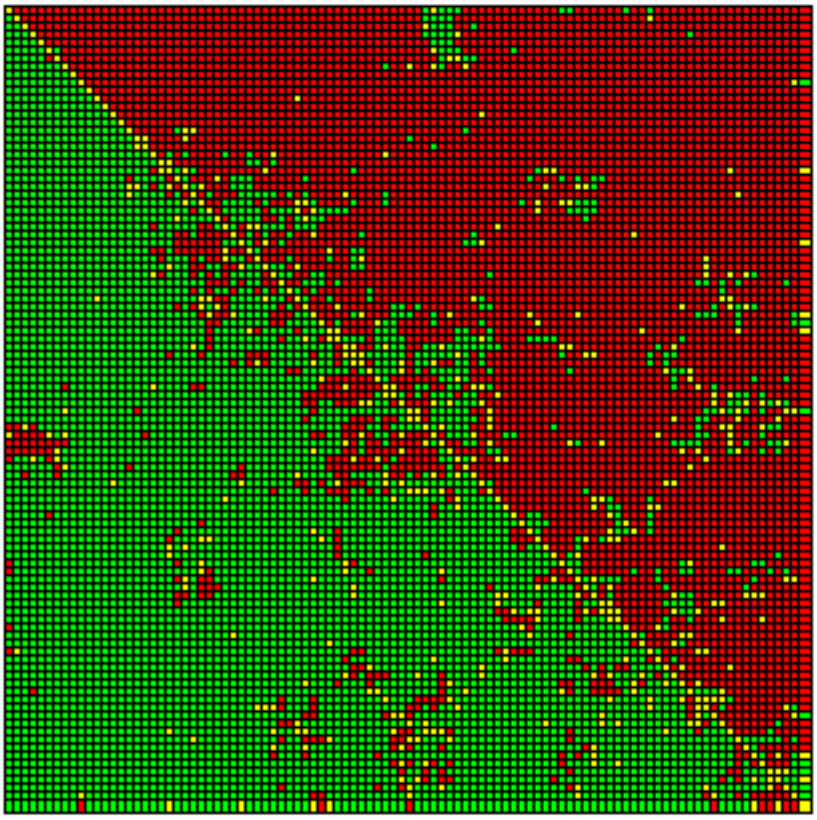

Figura 50: Resultado de 1000 simulaciones de un modelo de 100×100 con $p_H = 0{,}4$ y $p_V = 0{,}6$.

Como mencionamos anteriormente podemos colorear los vértices de acuerdo a una escala, asociando los colores de una gama a dicha escala.

Supongamos que queremos realizar una análisis similar al anterior pero en lugar de partir del vértice 1 deseamos partir del vértice central. Tomando $p_H = 0{,}25$ y $p_V = 0{,}75$ obtenemos un resultado como el de la figura 51. La gama elegida es de 5 tonos del color verde, correspondiendo el más claro al mayor valor de la escala. Nuevamente podemos ver que el resultado es el intuitivamente esperado. Esto es, la *nube* de vértices más alcanzados se ubica a lo largo del eje vertical que pasa por el centro de la grilla y su mayor concentración está alrededor del vértice central.

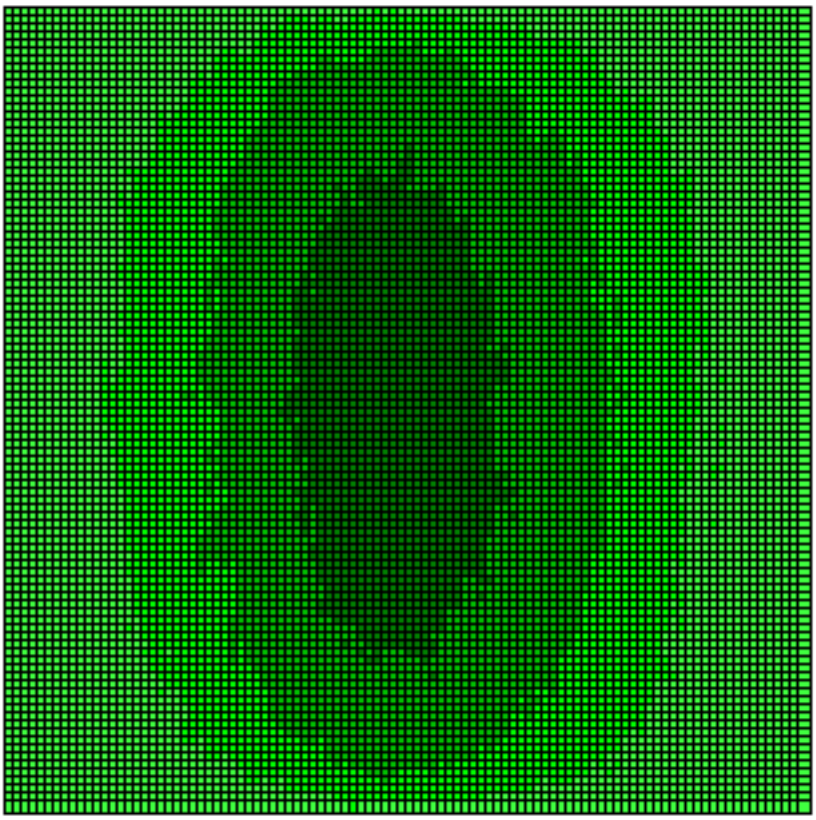

Figura 51: Resultado de 1000 simulaciones de un modelo de 100×100 con $p_H = 0{,}25$ y $p_V = 0{,}75$.

Si invertimos los valores de p_H y p_V obtenemos un resultado como el de la figura 52. Notemos que la nube se ubica a lo largo del eje horizontal.

Si aplicamos el coloreo de *espejo* con 3 colores sobre los últimos resultados

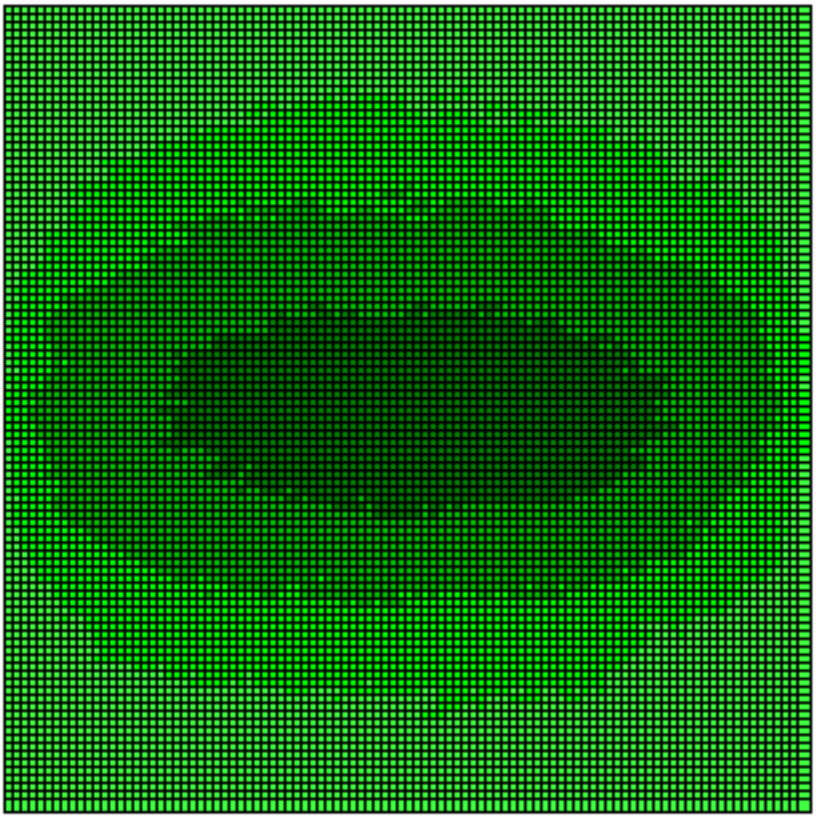

Figura 52: Resultado de 1000 simulaciones de un modelo de 100×100 con $p_H = 0{,}75$ y $p_V = 0{,}25$.

obtenemos un mapa como el de la figura 53. En él podemos ver que las cantidades asociadas a cada vértice comparadas con las de sus vértices *espejo* se distribuyen en forma casi simétrica alrededor de la diagonal principal.

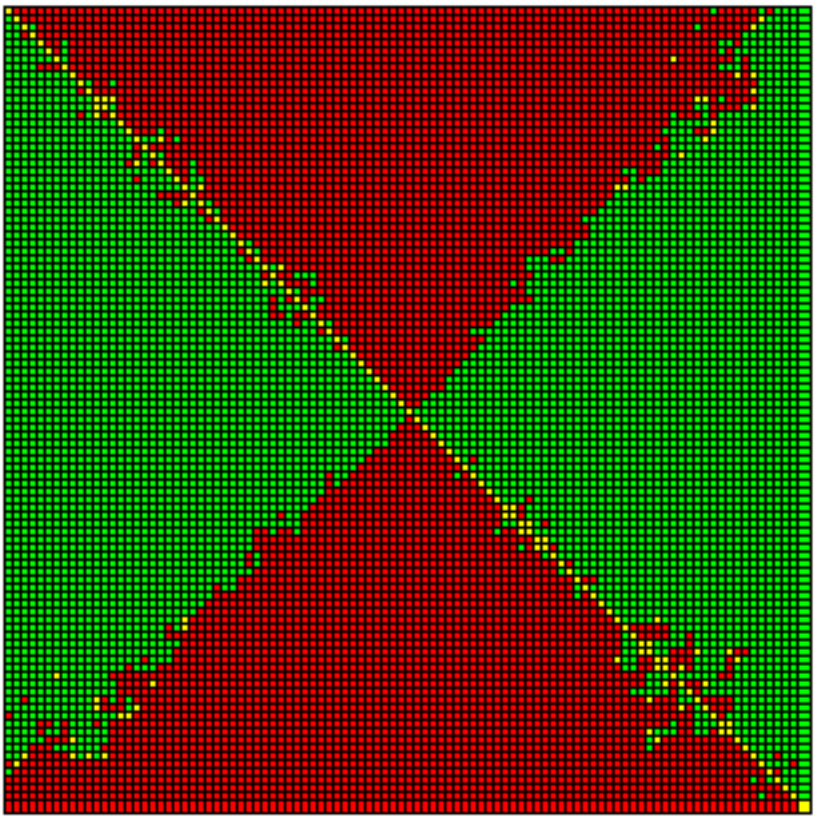

Figura 53: Resultado de 1000 simulaciones de un modelo de 100×100 con $p_H = 0{,}75$ y $p_V = 0{,}25$.

5.3. Modelos isotrópicos con enlaces a distancia 1 y a distancia k en $\mathbb{L}^2$

Estos son los modelos centrales de esta tesis y para los cuales no nos interesa conocer cuál es la probabilidad crítica dado k, sino analizar el impacto en la probabilidad crítica al variar k. Más precisamente, nos preguntamos: ¿cambia la probabilidad crítica al aumentar k?, y si cambia: ¿disminuye o aumenta al aumentar k?.

En primer lugar describamos la estructura de estos modelos e introduzcamos una notación para los mismos.

5.3.1. Notación $Square : (V : \{\ldots\}, H : \{\ldots\})$

Estos modelos puden ser vistos como una variación del modelo *square* al que se le agregan o quitan ejes. Por esta razón incluimos en su nombre el prefijo *Square*.

Recordemos que podemos describir el patrón de un grafo en términos del vecindario de cada vértice (si este es uniforme para todo vértice). En particular, podemos describir el modelo *square* diciendo que cada vértice esta conectado con los vértices a distancia 1 en sentido vertical y con los vértices a distancia 1 en el sentido horizontal.

Podemos elegir la siguiente notación para un modelo como el descripto:

$$Square : (V : \left\{\delta_1^v, \delta_2^v, \ldots, \delta_s^v\right\}, H : \left\{\delta_1^h, \delta_2^h, \ldots, \delta_t^h\right\})$$

donde el vecindario de un vértice cualquiera está descripto por los conjuntos V y H, siendo V un conjunto con las distancias de los vecinos en el sentido vertical (δ_i^v), y H un conjunto con las distancias de los vecinos en sentido horizontal (δ_i^h).

Por ejemplo, el modelo *square* podemos notarlo como $Square : (V : \{1\}, H : \{1\})$.

Los modelos estudiados entonces son los siguientes:

- $Square(V : \{1, k\}, H : \{1, k\})$
- $Square(V : \{1, k\}, H : \{1\})$
- $Square(V : \{1\}, H : \{1, k\})$
- $Square(V : \{k\}, H : \{1\})$
- $Square(V : \{1\}, H : \{k\})$

donde k es un número entero variable y cumple que $k \geq 1$. Observemos que para un modelo definido sobre una grilla de $N \times N$ no tiene sentido que sea $k > N$ ya que dichas conexiones no serían factibles.

5.3.2. Conjetura

Nuestra conjetura inicial era que para todos los modelos estudiados se cumple que: **la probabilidad crítica disminuye a medida que k aumenta**.

Dicho más formalmente, para los modelos enumerados en 5.3.1 se cumple que:

$$p_c(k+1) \leq p_c(k)$$

Observemos que este hecho no parece ser algo intuitivo ya que *las posibilidades de que se genere un cluster percolante a partir de un vértice cualquiera parecen ser las mismas independientemente del valor k*. Dicho más formalmente, si bien a partir de un nodo podemos acceder (con alguna probabilidad) a vértices más lejanos en un modelo $k+1$ que en un modelo k, nada parece indicar que esto influya sobre la probabilidad de que se generen clusters percolantes.

A continuación resumimos las simulaciones realizadas para cada uno de los modelos, variando k entre 2 y 15, sobre grillas de 1000×1000. Nuevamente, utilizamos el algoritmo explicado en 3.7 con 100 iteraciones, 10 evaluaciones por iteración y un porcentaje de aceptación del 80 %.

5.3.3. *Square*($V : \{1, k\}, H : \{1, k\}$)

Podemos ver gráficamente el vecindario de un vértice v dado en este modelo en la figura 54.

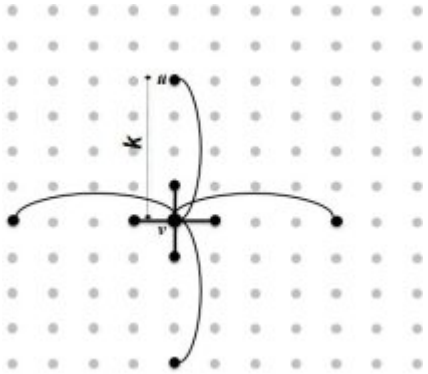

Figura 54: *Square*($V : \{1, k\}, H : \{1, k\}$).

En el cuadro 11 resumimos los resultados de las simulaciones para este modelo, donde podemos obervar un desplazamiento de la región crítica hacia la izquierda a medida que k aumenta. Sin embargo, notemos que el límite superior no desciende por debajo de 0,2.

2	0,21875	0,25
3	0,19375	0,2
4	0,18125	0,2
5	0,175	0,2
6	0,171875	0,2
7	0,16875	0,2
8	0,165625	0,2
9	0,165625	0,2
10	0,1640625	0,2
11	0,1640625	0,2
12	0,1625	0,2
13	0,1625	0,2
14	0,1625	0,2
15	0,1625	0,2

Cuadro 11: Regiones críticas del modelo bond *Square*($V : \{1, k\}, H : \{1, k\}$).

5.3.4. *Square*$(V : \{1, k\}, H : \{1\})$

En la figura 55 podemos ver gráficamente el vecindario de un vértice v dado en este modelo.

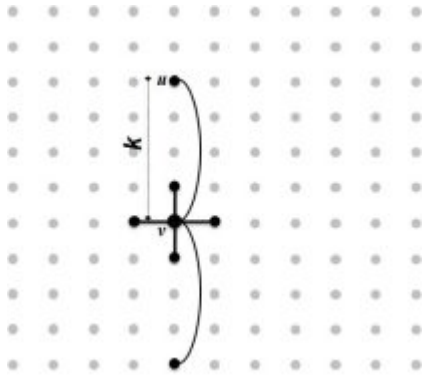

Figura 55: *Square*$(V : \{1, k\}, H : \{1\})$.

En el cuadro 12 resumimos los resultados de las simulaciones para este modelo. Nuevamente, podemos obervar un desplazamiento de la región crítica hacia la izquierda a medida que k aumenta, y también un *estancamiento* del límite superior en 0,275.

2	0,31875	0,35
3	0,29375	0,3
4	0,278125	0,3
5	0,26875	0,275
6	0,265625	0,275
7	0,2625	0,275
8	0,259375	0,275
9	0,25625	0,275
10	0,253125	0,275
11	0,2515625	0,275
12	0,25	0,275
13	0,25	0,275
14	0,24375	0,275
15	0,24375	0,275

Cuadro 12: Regiones críticas del modelo *Square*$(V : \{1, k\}, H : \{1\})$.

5.3.5. $Square(V : \{1\}, H : \{1, k\})$

En la figura 56 podemos ver gráficamente el vecindario de un vértice v dado en este modelo.

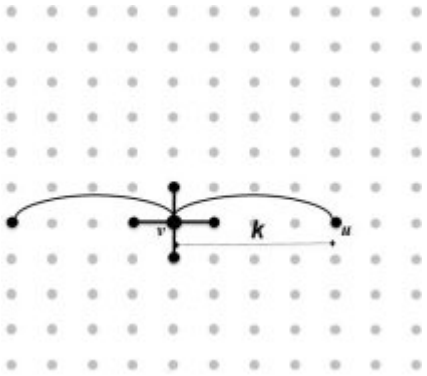

Figura 56: $Square(V : \{1\}, H : \{1, k\})$.

En el cuadro 13 resumimos los resultados de las simulaciones para este modelo. Aquí también podemos observar un desplazamiento hacia la izquierda a medida que k aumenta junto con un estancamiento del límite superior en 0,275.

2	0,325	0,35
3	0,3	0,325
4	0,2875	0,3
5	0,278125	0,3
6	0,275	0,3
7	0,26875	0,275
8	0,26875	0,275
9	0,265625	0,275
10	0,265625	0,275
11	0,265625	0,275
12	0,2625	0,275
13	0,2625	0,275
14	0,2625	0,275
15	0,2625	0,275

Cuadro 13: Regiones críticas del modelo $Square(V : \{1\}, H : \{1, k\})$.

5.3.6. *Square*($V : \{k\}, H : \{1\}$)

En la figura 57 vemos el vecindario de un vértice v dado en este modelo.

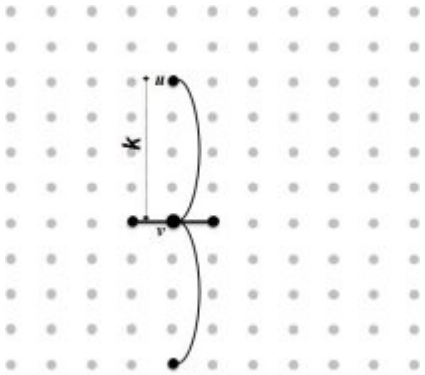

Figura 57: *Square*($V : \{k\}, H : \{1\}$).

En el cuadro 14 vemos los resultados de las simulaciones para este modelo. Obveservemos que este modelo equivale a un *cambio de escala* en el sentido vertical del modelo *square*. Por tal motivo, la probabilidad crítica no debería variar y debería ser 0,5. Sin embargo, vemos un desplazamiento hacia la izquierda de la región crítica. Este efecto se debe a dos factores combinados. En primer lugar mantenemos fijo tamaño de la grilla para todos los valores de k. Intuitivamente podríamos haber variado el tamaño de la grilla el sentido vertical para mantener la proporción con el tamaño horizontal (en términos de vértices accesibles). En segundo lugar, recordemos que estamos buscando percolación en el sentido vertical. Ambos factores tienen un impacto en el resultado de las simulaciones.

2	0,4875	0,5
3	0,4875	0,5
4	0,475	0,5
5	0,475	0,49375
6	0,475	0,49375
7	0,475	0,49375
8	0,475	0,4875
9	0,4625	0,4875
10	0,4625	0,4875
11	0,4625	0,4875
12	0,4625	0,475
13	0,4625	0,475
14	0,45625	0,475
15	0,45	0,4625

Cuadro 14: Regiones críticas del modelo *Square*($V : \{k\}, H : \{1\}$).

5.3.7. *Square*($V : \{1\}, H : \{k\}$)

En la figura 58 vemos el vecindario de un vértice v dado en este modelo.

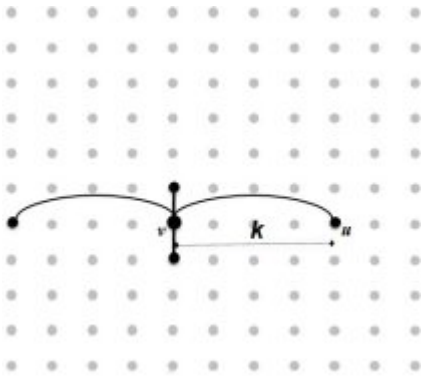

Figura 58: *Square*($V : \{1\}, H : \{k\}$).

En el cuadro 15 vemos los resultados de las simulaciones para este modelo. Al igual que para el modelo *Square*($V : \{k\}, H : \{1\}$), es intuitivo ver que este modelo es equivalente al modelo *square* realizando un cambio de escala en el sentido horizontal.

También en este caso notamos un desplazamiento de la región crítica pero, a diferencia del caso anterior, aquí el desplazamiento es hacia la derecha. Ocurre que para estas simulaciones también dejamos el tamaño de la grilla fijo y buscamos percolación en el sentido vertical. Estos factores combinados producen el efecto mencionado.

2	0,4875	0,5125
3	0,5	0,5125
4	0,5	0,5125
5	0,5	0,5125
6	0,5	0,5125
7	0,5	0,5125
8	0,50625	0,525
9	0,50625	0,5125
10	0,50625	0,525
11	0,50625	0,525
12	0,5125	0,525
13	0,5125	0,525
14	0,5125	0,525
15	0,5125	0,525

Cuadro 15: Regiones críticas del modelo $Square(V : \{1\}, H : \{k\})$.

5.4. Modelos isotrópicos en $\mathbb{L}^3$

Aquí abordamos brevemente las simulaciones de los modelos en 3 dimensiones. Los algoritmos utilizados son los mismos que los empleados para 2 dimensiones aunque naturalmente la complejidad temporal y espacial se ve afectada considerablemente. Esto se debe básicamente a que el tamaño de los grafos (en cantidad de vértices y ejes) definidos sobre una grilla de 3 dimensiones es considerablemente mayor que los definidos en las grillas de 2 dimensiones.

Es importante destacar aquí podemos utilizar las mismas herramientas utilizadas para los modelos en 2 dimensiones gracias al diseño (de objetos) de nuestro simulador, en el cual los diferentes conceptos fueron representados en forma simple y desacoplada del resto, y su interacción fue establecida mediante reglas claras. De hecho, es posible definir modelos de más dimensiones con solo definir nuevas grillas, aprovechando todos los conceptos representados hasta el momento como los diferentes tipos de patrones, las políticas de apertura, los algoritmos, etc..

A continuación resumimos los resultados de algunas simulaciones del modelo más básico en 3 dimensiones al que denominamos *cube*. Al igual que para las simulaciones anteriores, utilizamos el algoritmo para acotar la región crítica para 50 iteraciones, 10 evaluaciones por iteración y un porcentaje de aceptación del 80 %. También realizamos un análisis estadístico del comportamiento del estimador de $\theta(p)$.

5.4.1. Cube

En este modelo cada vértice se conecta con sus 6 vecinos a distancia unitaria. Podemos ver el vecindario de un vértice en la figura 59.

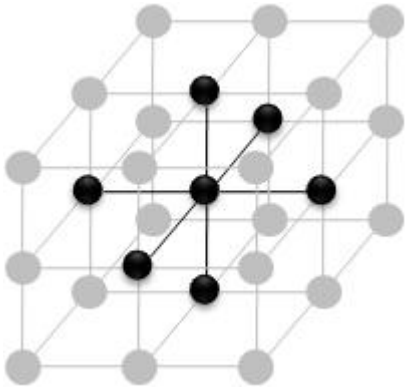

Figura 59: Vecindario de un vértice en un modelo con el patrón *cubic*.

Las probabilidades críticas para este modelo son 0,2488126 ([LOZ/98]) en el caso bond y 0,3116081 ([BFM/98]) en el caso site. En los cuadros 16 y 17 vemos los resultados de las simulaciones para ambos casos. En ellas podemos apreciar que las simulaciones se acercan a los valores críticos conocidos con bastante precisión.

Tamaño	Límite Inferior	Límite Superior
$10 \times 10 \times 10$	0,24414980	0,26713867
$20 \times 20 \times 20$	0,25078125	0,26063058
$30 \times 30 \times 30$	0,24853556	0,25579490
$40 \times 40 \times 40$	0,25006182	0,25302352
$50 \times 50 \times 50$	0,25007629	0,25234235
$60 \times 60 \times 60$	0,25000076	0,25246880
$70 \times 70 \times 70$	0,25000078	0,25125560
$80 \times 80 \times 80$	0,25000000	0,25143864
$90 \times 90 \times 90$	0,24814763	0,25187697
$100 \times 100 \times 100$	0,24811754	0,25115849

Cuadro 16: Regiones críticas del modelo bond *cube*.

Tamaño	Límite Inferior	Límite Superior
$10 \times 10 \times 10$	0,30673842	0,33701161
$20 \times 20 \times 20$	0,30792739	0,33196268
$30 \times 30 \times 30$	0,31292802	0,32329063
$40 \times 40 \times 40$	0,30964785	0,32186927
$50 \times 50 \times 50$	0,31328814	0,31865127
$60 \times 60 \times 60$	0,31250089	0,31798963
$70 \times 70 \times 70$	0,31250201	0,31533196
$80 \times 80 \times 80$	0,31250015	0,31680994
$90 \times 90 \times 90$	0,31103540	0,31600640
$100 \times 100 \times 100$	0,30985436	0,31464378

Cuadro 17: Regiones críticas del modelo site *cube*.

6. Conclusiones

En este trabajo abordamos brevemente la teoría de la percolación y sus aplicaciones, realizando algunas simulaciones para modelos conocidos, para luego centrarnos en el estudio de algunos modelos específicos que consideramos de interés. Dichos modelos se caracterizan por tener una estructura en la que sus vértices se conectan con vecinos a distancia unitaria, tanto en sentido vertical como horizontal, como a distancia fija k. Al mismo tiempo, pueden ser isotrópicos o bien anisotrópicos.

El foco de interés nuestro sobre estos modelos no fue su probabilidad crítica sino entender el comportamiento en función de la variable k. Sobre este punto planteamos varias conjeturas iniciales que entendemos como novedosas en el campo de la percolación, y que luego buscamos verificar mediante simulaciones números. Para algunos modelos encontramos un desplazamiento de la región crítica hacia el cero, aunque con un estancamiento del límite superior a partir de cierto valor de k; mientras que para otros modelos las simulaciones evidenciaron la invalidez de la conjetura. Para estos últimos encontramos una justificación basada en el hecho de que son cambios de escala del modelo *square*, para el cual se conoce la probabilidad crítica, y el efecto producido por el cambio de proporciones en las simulaciones numéricas.

Al mismo tiempo hemos desarrollado una herramienta lo suficientemente flexible como para permitirnos estudiar todos los modelos detallados a lo largo del presente informe, con un desempeño más que aceptable. Adicionalmente, esta herramienta brinda la posiblidad de crear una gran variedad de nuevos modelos mediante la combinación de patrones existentes o la creación de nuevos patrones (gráficamente), con diferentes tipos de grillas, políticas de apertura y varios parámetros más. Creemos que la flexibilidad lograda reposa en un diseño de objetos de baja complejidad, es decir, con objetos simples, con una responsabilidad acotada y con reglas claras en su interacción.

Si bien consideramos de gran valor los resultados números obtenidos

para los modelos estudiados, así como la herramienta desarrollada, lo que creemos merece ser destacado por sobre dichos resultados es que a la hora de enfrentarnos a una problemática dada, en nuestro caso la simulación de procesos de percolación, el hecho de enfocarnos en representar computacionalmente tal problemática, en lugar de poner el acento en escribir algoritmos particulares para los modelos que nos interesaban, nos condujo no solamente a resultados valiosos sino a nuevos interrogantes y problemas por estudiar. Esta reflexión no es novedosa ya que es un enfoque perseguido dentro del paradigma de objetos. Sin embargo, encontramos nuestro trabajo un ejemplo práctico de tal enfoque, y por sobre todo, sentimos que es mediante este enfoque donde se realza el sentido de utilizar la computadora como medio de aprendizaje y enriquecimiento de nuestro conocimiento sobre un aspecto de la realidad, y no como una simple calculadora (extremadamente) veloz. Tal es así, que durante el proceso de desarrollo y de pruebas nos planteamos muchas variantes de los modelos estudiados, que por cuestiones de tiempo preferimos dejar su estudio para trabajos futuros.

7. Bibliografía

Referencias

[BFM/98] Ballesteros, P. N., Fernández, L. A., Martín-Mayor, V., Muñoz, A., Sudepe, Parisi, G., Ruiz-Lorenzo, J. J.: *Scaling corrections: site percolation and Ising model in three dimensions*, Journal of Physics A 32: 1-13, 1998.

[BRO/93] Bedrikovetsky, P., Rowan, G.: *Mathematical Theory of Oil and Gas Recovery: With Applications to Ex-USSR Oil and Gas Fields*, Springer, 1993.

[BHA/57] Broadbent, S. R., Hammersley, J. M.: *Percolation processes I. Crystals and mazes*, Proceedings of the Cambridge Philosophical Society 53, 629-641, 1957.

[CFA/00] Favier, C.: *Percolation model of fire dynamic*, Service de Physique de lÉtat Condensé, CEA Saclay-Orme des Merisiers, F-91191 Gif-sur-Yvette, France, 2000.

[FDB/08] Feng, X., Deng, Y., Blöte, H. W. J.: *Percolation transitions in two dimensions*, Physical Review E 78: 031136, 2008.

[FSI/04] Fontes, L. R. G., Sidoravicius, V.: *Percolation*, in: G. F. Lawler (ed.), School and conference on probability theory, The Abdus Salam International Center for Theoretical Physics, Trieste, 101-201, 2004.

[GPA/94] Guéguen, Y., Palciauskas, V.: *Introduction to the Physics of Rocks*, Princeton University Press, 1994.

[GRI/99] Grimmett, G.: *Percolation, Second Edition*, Springer, Verlag Berlin Heidelberg, 1999.

[HAR/60] Harris, T. E.: *A lower bound for the critical probability in a certain percolation process*, Proceedings of the Cambridge Philosophical Society 56, 1960.

[KES/80] Kesten, H.: *The critical probability of bond percolation on the square lattice equals* $\frac{1}{2}$, Communication in Mathematical Physics 74, 1980.

[LIM/04] Lima, B. N. B. de, Procacci, A.: *A note on anisotropic percolation*, Letters in Mathematical Physics, vol. 70, issue 3, 223-229, 2004.

[LOZ/98] Lorenz, C. D., Ziff, R. M.: *Precise determination of the bond percolation thresholds and finite-size scaling corrections for the sc, fcc, and bcc lattices*, Physical Review E 57: 230-236, 1998.

[MTW/73] McCoy, B. M., Wu, T. T.: *The Two-dimensional Ising Model*, Harvard University Press, 1973.

[MSV/01] Menshikov, M., Sidoravicius, V., Vachkovskaia, M.: *A note on two-dimensional truncated long-range percolation*, Advances in Applied Probability 33, 912-929, 2001.

[MNE/00] Moore, C., Newman, M. E. J.: *Epidemics and percolation in small-world networks*, Phys. Rev. E 61, 5678 - 5682, 2000. (http://www.santafe.edu/research/publications/workingpapers/00-01-002.pdf)

[NTD/00] Nahmias, J., Téphany, H., Duarte, J., Letaconnoux, S.: *Fire spreading experiments on heterogeneous fuel beds. Applications of percolation theory*, Can. J. For. Res. 30(8): 1318-1328, 2000.

[NEZ/00] Newman, M. E. J., Ziff, R. M.: *Efficient Monte-Carlo algorithm and high-precision results for percolation*, Physical Review Letters 85 (19): 4104-4107, 2000.

[PM/98] Park, S. K., Miller, K. W.: *Random Number Generators: Good Ones Are Hard to Find*, Comm. Asso. Comp. Mach., 31(10):1192–1201, 1988.

[SKE/98] Sykes, M. F., Essam, J. W.: *Exact critical percolation probabilities for site and bond problems in two dimensions*, Journal of Mathematical Physics (N.Y.) 5 (8): 1117-1127, 1964.

[VDM/97] Van der Marck, S. C.: *Percolation thresholds and universal formulas*, Physical Review E 55: 1514-1517, 1997.

[WIE/84] Wierman, J. C.: *A bond percolation critical probability determination based on the star-triangle transformation*, Journal of Physics A 17: 1525-1530, 1984.

[ZCL/01] Zekri, N., Clerc, J.: *Statistical and Dynamical Study of Disease Propagation in a Small World Network*, 2001. (http://arxiv.org/PS_cache/cond-mat/pdf/0107/0107562v3.pdf)

[ZIS/99] Ziff, R. M., Suding, P. N.: *Site percolation thresholds for Archimedean lattices*, Physical Review E 60 (1): 275-283, 1999.