Universidad Nacional Autónoma de México

Facultad de Ciencias

Descontaminación de Hipercubos por Agentes Móviles

T E S I S

QUE PARA OBTENER EL TÍTULO DE:
ACTUARIA

PRESENTA:
ARIANA PAOLA CORTÉS ÁNGEL

DIRECTOR DE TESIS:
CÉSAR HERNÁNDEZ CRUZ

Ciudad Universitaria, Cd. Mx., 2017

1.Datos del alumno

Apellido paterno	Cortés
Apellido materno	Ángel
Nombre(s)	Ariana Paola
Teléfono	78 25 97 69
Universidad	Universidad Nacional Autónoma de México
Facultad o escuela	Facultad de Ciencias
Carrera	Actuaría
Número de cuenta	303671871

2. Datos del tutor

Grado	Dr.
Nombre(s)	César
Apellido paterno	Hernández
Apellido materno	Cruz

3. Datos del sinodal 1

Grado	Dra.
Nombre(s)	Mucuy Kak del Carmen
Apellido paterno	Guevara
Apellido materno	Aguirre

4. Datos del sinodal 2

Grado	Dr.
Nombre(s)	Jesús
Apellido paterno	Alva
Apellido materno	Samos

5. Datos del sinodal 3

Grado	M. en C.
Nombre(s)	Patricio Ricardo
Apellido paterno	García
Apellido materno	Vázquez

6. Datos del sinodal 4

Grado	Dra.
Nombre(s)	Ana Paulina
Apellido paterno	Figueroa
Apellido materno	Gutiérrez

7.Datos del trabajo escrito.

Título	Descontaminación de Hipercubos por Agentes Móviles
Número de páginas	54 p.
Año	2017

Índice general

Introducción

La descontaminación en gráficas es un tema que ha sido estudiado desde la década de los setentas [13], a lo largo de este tiempo también se le ha conocido como "búsqueda en gráficas", "persecución y evasión en gráficas", entre otros nombres. Asimismo, se han estudiado diversas variantes de este problema, incluídas las versiones por vértices y aristas, o bien, imponiendo restricciones a la estructura de la gráfica [1, 7] así como factores externos a la misma, por ejemplo condicionando la movilidad o la cantidad de los agentes [2, 5], o incluso considerando variantes de inmunidad [12, 8].

En el presente trabajo, el objeto de estudio es la descontaminación de hipercubos n-dimensionales, para los cuales analizaremos tres estrategias de descontaminación:

- Limpieza con Coordinador;
- Limpieza sin Coordinador; y
- Limpieza sin Coordinador y con Clonación.

Cuando hablamos de descontaminar un hipercubo nos referimos a que todos los vértices de éste se encontrarán limpios y/o resguardados al terminar la desinfección. Un vértice se encuentra resguardado siempre que haya al menos un agente en él y se dice limpio si ya pasó un agente por él de forma que ahora su vecindad está resguardada o limpia.

Para la estrategia de limpieza con coordinador utilizaremos el Algoritmo de Limpieza con Localidad, éste se caracteriza por necesitar un coordinador, el cual guiará a todos sus agentes vértice por vértice, ya que entre agentes no puede haber comunicación ni hay forma de que puedan ver el estado de sus vértices vecinos. Por medio de este algoritmo llegaremos a la cota inferior óptima de agentes.

En la estrategia de limpieza sin coordinador nos conduciremos por medio del Algoritmo de Limpieza con Visibilidad, en este algoritmo no usamos un coordinador pues los agentes son autónomos, es decir, que pueden decidir a donde moverse solos, pues tienen la capacidad de ver el estado de sus vértices vecinos, lo cual observaremos optimiza el tiempo de ejecución

Finalmente para la estrategia de limpieza sin coordinador y con clonación ejecutaremos dos algoritmos distintos, Algoritmo de Limpieza con Visibilidad y Clonación y el Algoritmo de Limpieza con Clonación y Sincronicidad, con el fin de llegar al mismo resultado, aunque ambos cuentan con propiedades diferentes, sus cualidades llegan a ser las mismas, ya que ambos van a tener agentes autónomos, su complejidad será óptima tanto en movimientos como en tiempo de ejecución y además utilizarán el mismo número de agentes $(n/2)$ durante el proceso.

De cada estrategia obtendremos el número de agentes, el número de movimientos y la cantidad de tiempo empleado por el algoritmo correspondiente, es decir, que veremos la eficiencia de cada estrategia tanto en tiempo como en número de recursos, para que en determinado momento según las necesidades con las que nos enfrentemos podamos seleccionar la estrategia que mejor nos funcione.

Capítulo 1

Preliminares

1.1. Gráficas

Los conceptos, notación y resultados presentados en esta sección pueden ser consultados en [3, 6].

Una ***gráfica***, G, es una terna ordenada $(V(G), E(G), \psi_G)$, compuesta por un conjunto de vértices, $V(G)$, un conjunto de aristas, $E(G)$, los cuales son ajenos entre si, y una función de adyacencia, ψ_G, que le asocia a cada arista de G un par no ordenado de vértices (no necesariamente distintos). Si e es una arista y u y v son vértices tales que $\psi_G(e) = \{u, v\}$, entonces se dice que e une a u y a v y los vértices u y v se llaman ***extremos*** de e. Para denotar a la pareja no ordenada $\{u, v\}$ escribiremos uv. Dos vértices que son incidentes con una arista en común se dice que son ***adyacentes*** o ***vecinos***, así como dos aristas que son incidentes con un vértice en común. Al conjunto de vértices adyacentes o vecinos a un vértice v le llamamos ***vecindad*** de v y la denotamos $N(v)$. Al número de vértices en G se le denota $v(G)$ y se le llama ***orden*** de G, mientras que al número de aristas en G se le denota $e(G)$ y se le llama ***tamaño*** de G. En la Figura 1.1 podemos observar el ejemplo de una gráfica $G = (V(G), E(G))$, donde $V(G) = \{v_1, v_2, v_3, v_4\}$, $E(G) = \{e_1, e_2, e_3, e_4, e_5, e_6\}$, $\psi_G(e_1) = v_1v_2$, $\psi_G(e_2) = v_2v_3$, $\psi_G(e_3) = v_1v_3$, $\psi_G(e_4) = v_2v_4$, $\psi_G(e_5) = v_4$ y $\psi_G(e_6) = v_1v_3$.

Una gráfica H es llamada ***subgráfica*** de la gráfica G si $V(H) \subseteq V(G)$, $E(H) \subseteq E(G)$ y la función ψ_H es la restricción de ψ_G a las aristas de H, es decir, $\psi_H = \psi_G \restriction_{E(H)}$, lo anterior se denota por $H \subseteq G$, además se dice que G contiene a H.

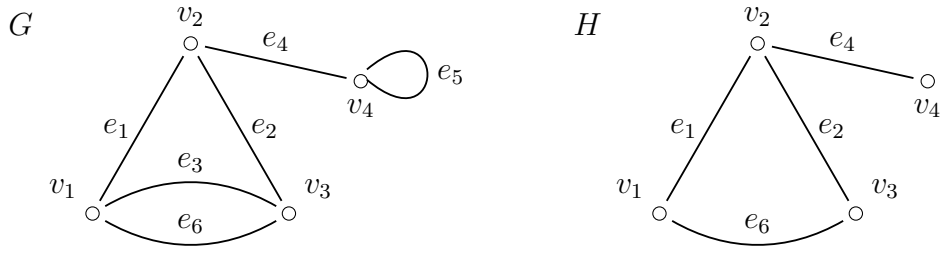


Figura 1.1: Ejemplo de dos gráficas G y H, tales que $H \subseteq G$.

En la Figura 1.1 se muestran dos gráficas G y H tales que H es subgráfica de G. Si $H \subseteq G$ y además se tiene que $V(H) = V(G)$, entonces H es llamada una subgráfica ***generadora*** de G . Dado $X \subseteq V(G)$, la subgráfica de G ***inducida*** por X es la gráfica $G[X]$ tal que $V(G[X]) = X$ y donde para cualquier par de vértices $u, v \in X$ se tiene que $uv \in E(G[X])$ si y sólo si $uv \in E(G)$.

Un ***lazo*** es una arista que tiene sus dos extremos iguales. Una ***gráfica vacía*** es aquella que no tiene aristas. Cuando la gráfica tiene un único vértice y ninguna arista decimos que es una gráfica ***trivial*** y en cualquier otro caso decimos que es ***no trivial***. Una ***gráfica finita*** es aquella en la que los conjuntos de vértices y aristas son finitos. Decimos que una gráfica es ***simple*** si aristas distintas tienen extremos distintos y además no tiene lazos.

En este trabajo nosotros sólo nos referiremos a gráficas simples y a gráficas finitas, entonces a partir de este punto siempre que mencionemos una gráfica nos estaremos refiriendo a una gráfica simple y finita. Por tanto observemos que para que H sea subgráfica de G basta ver que $V(H) \subseteq V(G)$ y $E(H) \subseteq E(G)$ (observemos que ya que son gráficas simples podemos omitir a la función de adyacencia y pensar a G como una pareja $G = (V, E)$ donde V es un conjunto distinto al vacío y $E \subseteq [V]^2 = \{S \subseteq V \mid |S| = 2\}$, es decir no habrán lazos, ni aristas distintas con los mismos extremos).

Decimos que dos gráficas G y H son ***idénticas*** si $V(G) = V(H)$ y $E(G) = E(H)$, y lo denotamos por $G = H$. Dos gráficas G y H son ***isomorfas***, $G \cong H$, si existe una función biyectiva $\rho : V(G) \to V(H)$ tal que $xy \in E(G)$ si y sólo si $\rho(x)\rho(y) \in E(H)$.

Una ***gráfica completa*** es una gráfica simple en la cual cualesquiera dos vértices distintos están unidos por una arista. Una ***gráfica bipartita*** es aquella en la cual sus vértices se pueden dividir en dos conjuntos X y Y, ajenos y no vacíos, de forma que cada arista tiene un extremo en X y otro en Y. Entonces (X, Y) se llama bipartición

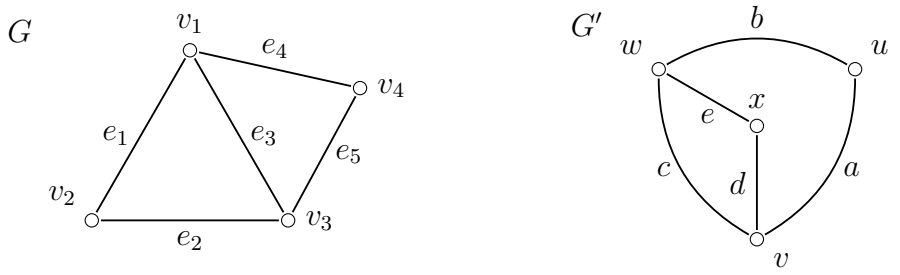


Figura 1.2: Podemos observar que la gráfica G es isomorfa a la gráfica G'.

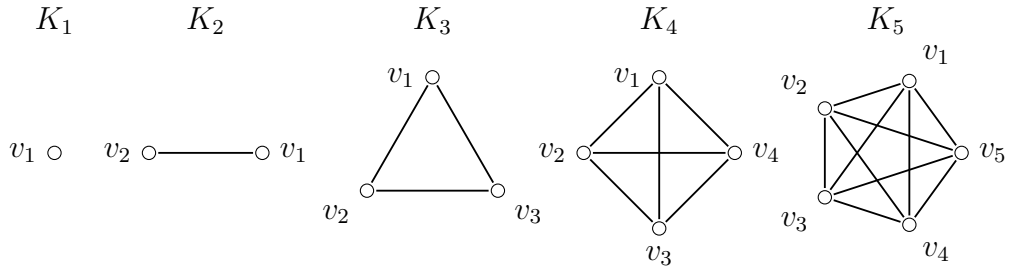


Figura 1.3: Gráficas completas de orden menor o igual a 5.

de la gráfica. Una ***gráfica bipartita completa*** es una gráfica bipartita simple con una bipartición (X, Y) en la que cada vértice de X se une con cada vértice de Y. Si $|X| = m$ y $|Y| = n$ entonces esta gráfica bipartita completa se denota $K_{m,n}$.

Un ***camino*** en G es una sucesión finita $W = v_0e_1v_1e_2v_2 \dots e_kv_k$ de vértices y aristas alternadas. Para cada $1 \leq i \leq k$, los extremos de e_i son v_{i-1} y v_i y decimos que W es un camino de v_0 a v_k. Llamamos a v_0 y a v_k inicio y término de W respectivamente, a los demás vértices de W, $v_1, v_2, \dots, v_{k-1}$, los conocemos como vértices internos. La ***longitud*** de W es el entero k. En una gráfica simple un camino $W = v_0e_1v_1e_2v_2 \dots e_kv_k$ se determina por una sucesión $v_0, v_1, \dots, v_k$ de los vértices de W, por tanto un camino en una gráfica simple se puede expresar simplemente por su sucesión de vértices, a saber $W = v_0v_1v_2 \dots v_k$. Si las aristas $e_1, e_2, e_3, \dots, e_k$ de un camino son distintas entonces decimos que es un ***paseo*** . Si los vértices $v_1, v_2, v_3, \dots, v_k$ de un camino son distintos entonces la llamamos ***trayectoria***, donde v_1 y v_k son los extremos de la trayectoria. Un ***camino cerrado*** $\mathscr{C}$, es un camino en el cual los extremos son el mismo vértice, $\mathscr{C} = v_0v_1 \dots v_k$ donde $v_k = v_0$. Un ***ciclo*** es un camino cerrado que no repite vértices salvo el primero y el último.

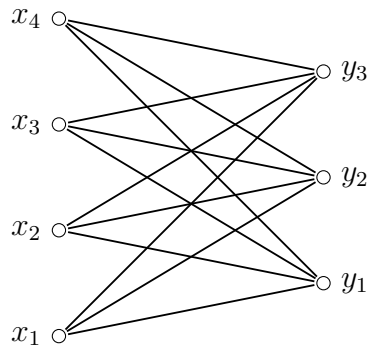


Figura 1.4: Gráfica bipartita completa $K_{4,3}$.

Dado un camino $W = v_0v_1v_2\dots v_k$ y dados $0 \leq i \leq j \leq k$, definimos a la v_iv_j-***sección*** de W, denotada v_iWv_j, como el subcamino de W tal que $v_iWv_j = v_iv_{i+1}\dots v_{j-1}v_j$.

Dados $x, y \in V(G)$ una *xy-trayectoria* en G es una trayectoria en G con extremos x, y. La ***distancia*** entre x y y es la longitud de la xy-trayectoria más corta en G y se denota $d_G(x, y)$, cuando no haya lugar a confusiones lo escribiremos $d(x, y)$. El ***diámetro*** de G es la distancia máxima entre dos vértices de G.

Por otro lado, una gráfica G es llamada ***conexa*** si para todo par de vértices $x, y \in V(G)$ existe un *xy-camino* en G. Una componente conexa es una subgráfica conexa máxima por contención.

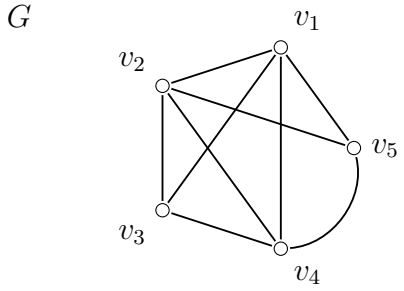


Figura 1.5: Una gráfica G donde podemos ver el camino $v_3v_1v_4v_2v_5v_1v_4v_2$, el paseo $v_1v_3v_2v_1v_4$, la trayectoria $v_1v_4v_5v_2v_3$, el camino cerrado $v_1v_3v_2v_1v_5v_4v_1$ y el ciclo $v_4v_5v_1v_2v_4$.

El ***grado*** de un vértice v, $d_G(v)$, en una gráfica G es el número de aristas de G que inciden en v. Denotamos al grado mínimo de G como $\delta(G)$ y al grado máximo de G, $\Delta(G)$.

Proposición 1.1.1. *Sea G una gráfica, todo xy-camino en G contiene una xy-trayectoria en G.*

Demostración. Por inducción sobre la longitud del camino. Cuando la longitud del camino W es 1, tenemos una arista en W. Cuando la longitud del camino W es n, suponemos que hay una xy-trayectoria en W. Sea $\mathscr{C} = (x = v_0, v_1, \ldots, v_n, v_{n+1} = y)$ un camino de longitud $n+1$, y consideremos a $\mathscr{C}' = (x, v_1, \ldots, v_n)$, un camino de longitud n que por hipótesis inductiva ya contiene una xv_n-trayectoria, P. Ahora veamos que si P contiene a y quiere decir que existe una xy-trayectoria en $P \subseteq \mathscr{C}' \subseteq \mathscr{C}$. Si P no contiene a y entonces $P \cup v_n y \subseteq \mathscr{C}$ es una xy-trayectoria. ■

Proposición 1.1.2. *Si G es una gráfica en la que existen dos uv-trayectorias distintas entonces la unión de estas trayectorias contiene un ciclo.*

Demostración. Sean $P_1 = (u = x_1, \ldots, x_m = v)$, y $P_2 = (u = y_1, \ldots, y_n = v)$ dos uv-trayectorias distintas, consideremos a $r = \text{mín}\{i \in \mathbb{N} \mid x_i \neq y_i\}$, dicho r existe pues $P_1 \neq P_2$, y tomemos a $s = \text{mín}\{j \in \mathbb{N} \mid j > r, x_j \in P_2\}$, es decir $x_s = y_t$ para algún $r < t \leq n$, observemos que s existe puesto que $x_m = y_n = v$.

Por lo tanto $x_{r-1}P_1x_s$ y $y_{r-1}P_2y_t$ son dos trayectorias que se intersectan solamente en sus extremos, entonces $x_sP_1x_{r-1} \cup y_{r-1}P_2y_t = (x_s, \ldots, x_{r-1} = y_{r-1}, \ldots, y_t = x_s)$ es un ciclo. ■

Para demostrar la próxima proposición utilizaremos el siguiente lema.

Lema 1.1.3. *Todo camino cerrado de longitud impar contiene un ciclo de longitud impar.*

Demostración. Demostración por inducción sobre el número de vértices que se repiten en el camino. Sea $W = (v_1, v_2, \ldots, v_k)$, un camino cerrado de longitud impar el cual no repite vértices, entonces W es un ciclo de longitud impar. Supongamos que para todo camino cerrado de longitud impar con k vértices repetidos, donde $0 \leq k \leq n$, contiene un ciclo de longitud impar. Sea $\mathscr{C}$ un camino cerrado de longitud impar con n vértices repetidos, $\mathscr{C} = (x_1, x_2, \ldots, x_r)$. Sin pérdida de generalidad existe j tal que $1 < j < r$ con $x_1 = x_j$, lo que nos permite construir dos caminos cerrados $\mathscr{C}_1$ y $\mathscr{C}_2$, $\mathscr{C}_1 = (x_1, \ldots, x_j)$ y $\mathscr{C}_2 = (x_j, \ldots, x_r)$. De estos dos caminos, uno tiene longitud par y otro longitud impar, además de que en cada uno se repiten menos de n vértices. Entonces, sin pérdida de generalidad supongamos que la longitud de $\mathscr{C}_1$ es impar, por hipotesis inductiva sabemos que contiene un ciclo de longitud impar, que a su vez está contenido en $\mathcal{C}$. ■

Proposición 1.1.4. *Si G es una gráfica con al menos dos vértices, entonces es bipartita si y sólo si no contiene ciclos impares.*

Demostración. Sea G una gráfica bipartita y $\mathscr{C} = (x_1, x_2, \ldots, x_r)$ un ciclo en G. Sea (V_1, V_2) la bipartición de $V(G)$, y supongamos sin pérdida de generalidad que $x_1 \in V_1$. Como $x_1x_2 \in E(G)$, por definición de gráfica bipartita $x_2 \in V_2$. Como $x_2x_3 \in E(G)$, por definición de gráfica bipartita $x_3 \in V_1$. Procediendo con un argumento inductivo, se puede verificar que, en general, $x_{2i-1} \in V_1$ y $x_{2i} \in V_2$ para $1 \leq i \leq \lceil r/2 \rceil$. Finalmente, al tener que $x_rx_1 \in E(G)$, se sigue que $x_r \in V_2$, por lo que r es par y por tanto $\mathscr{C}$ es par. Así, G no contiene ciclos de longitud impar.

Ahora, supongamos sin pérdida de generalidad que G es conexa y sea $x \in V(G)$. Definamos a los conjuntos

$$N_1 = \{y \in V(G) \text{ tal que } d_G(x, y) \text{ es par}\},$$

$$N_2 = \{y \in V(G) \text{ tal que } d_G(x, y) \text{ es impar}\}.$$

Observemos que $N_1 \cup N_2 = V(G)$, $N_1 \cap N_2 = \varnothing$, $N_1 \neq \varnothing \neq N_2$. Supongamos que existe $uw \in E(G)$ con extremos en N_1, es decir, $u \in N_1$ y $w \in N_1$, entonces la distancia de x a u y de x a w es par. Sea $\mathscr{C}_1$ una xu-trayectoria de longitud par y $\mathscr{C}_2$

una xw-trayectoria de longitud par veamos entonces que $\mathscr{C}_1 \cup uw \cup \mathscr{C}_2^{-1}$ es un camino cerrado de longitud impar, entonces, por el Lema 1.1.3 hay un ciclo de longitud impar, por lo tanto hay una contradicción, ya que G no contiene ciclos impares. Un razonamiento análogo cubre el caso en el que tomamos a $uw \in E(G)$ con extremos en N_2, ya que consideramos dos caminos de longitud impar, y al agregar la arista uw obtenemos un camino cerrado de longitud impar. Así, nuevamente hay un ciclo de longitud impar en G, resultando en una contradicción. ■

Una gráfica ***acíclica*** es aquella que no contiene ciclos. Un ***árbol*** es una gráfica conexa acíclica.

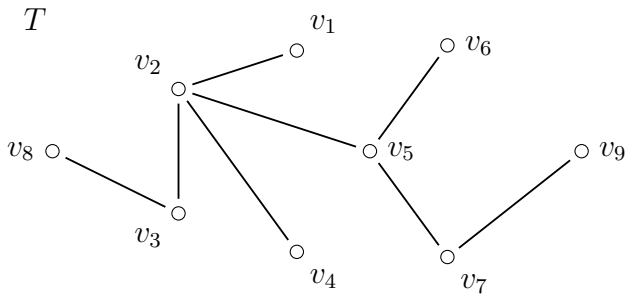


Figura 1.6: Ejemplo de un árbol T, gráfica conexa acíclica.

Teorema 1.1.5. *En un árbol T, cualesquiera dos vértices estan conectados por una xy-trayectoria única en T.*

Demostración. Por contradicción. Supongamos que existen $x, y \in V(T)$ y que existen P_1 y P_2 xy-trayectorias (distintas entre si) en T. Sabemos que existe $e = v_1v_2$, $e \in E(P_1)$ tal que $e \notin E(P_2)$, Podemos ver que $P_1 \cup P_2 - e$ es conexa, entonces existe una v_1v_2-trayectoria P_3 y entonces $P_1 \cup P_2$ contiene un ciclo, $P_3 \cup \{e\}$, pero $P_1 \cup P_2$ es subgráfica de T, y T es acíclica, por tanto es una contradicción. ■

Si T es un árbol, una ***hoja*** de T es un vértice $v \in V(T)$ tal que $d(v) = 1$.

Teorema 1.1.6. *Todo árbol no trivial tiene al menos dos hojas.*

Demostración. Sea T un árbol no trivial, entonces $d_T(v) \geq 1$ para todo $v \in V(T)$. Sea P una xy-trayectoria de longitud máxima en T, con extremos x y y, T no trivial. Afirmamos que x y y son hojas, pues de no ser así, suponemos sin pérdida de generalidad que $d_T(v) \geq 2$, entonces existe $w \in N_T(x)$ tal que no está en P. Por lo cual $\{wx\} \cup P$ es una trayectoria de mayor longitud que P, o existe $w \in N_T(x)$ que está en P y en consecuencia existe un ciclo, contradiciendo que T es acíclica. ■

Teorema 1.1.7. *Si T es un árbol, entonces $|E(T)| = |V(T)| - 1$.*

Demostración. Procederemos por inducción sobre $|V(T)|$, el orden de T. Si T tiene un solo vértice, el resultado se cumple trivialmente. Supongamos el resultado válido para todo árbol con a lo más n vértices. Sean T un árbol con $n+1$ vértices y uv una arista de T. Por el Teorema 1.1.5, no existe una uv-trayectoria en $T - uv$ y por lo tanto $T - uv$ tiene dos componentes conexas T_1 y T_2. Como T_1 y T_2 son acíclicas, resultan ser árboles, y por hipótesis inductiva $|E(T_i)| = |V(T_i)| - 1$, con $i \in \{1, 2\}$, de lo cual se sigue que

$$|E(T)| = |E(T_1)| + |E(T_2)| + 1 = |V(T_1)| + |V(T_2)| - 1 = |V(T)| - 1.$$

■

1.2. Hipercubos

Si G y H son gráficas, el ***producto cuadro o cartesiano***, $G\square H$, se define como la gráfica que tiene por conjunto de vértices al producto cartesiano $V(G) \times V(H)$ y donde

$$(g,h)(g',h') \in E(G\square H) \text{ si y sólo si } \begin{cases} g = g' \text{ y } hh' \in E(H) \\ \text{ó} \\ gg' \in E(G) \text{ y } h = h'. \end{cases}$$

A partir de la definición de producto cuadro podemos hacer una observación muy útil. Fijemos $g \in V(G)$ y llamemos H_g a la subgráfica de $G\square H$ inducida por el conjunto $\{(g,h)\colon\ h \in V(H)\}$. Como la primera coordenada de todos los vértices en H_g es g, resulta claro que $H_g \cong H$; el isomorfismo está dado por la función $\varphi(g,h) = h$. Análogamente, si fijamos $h \in V(H)$ y llamamos G_h a la subgráfica de

$G\square H$ inducida por el conjunto $\{(g,h)\colon\ g \in V(G)\}$, entonces $G_h \cong G$. Así, por cada vértice de H, tenemos una copia isomorfa de G y por cada vértice de G tenemos una copia isomorfa de H en $G\square H$.

Esta observación puede generalizarse para el caso en el que tengamos el producto cuadro de un número finito de gráficas. Denotaremos por $\prod_{i=1}^{n} G_i$ al producto cuadro de una familia de gráficas $\{G_1, \ldots, G_n\}$. Se sigue de la definición de producto cuadro que $(u_1, \ldots, u_n)(v_1, \ldots, v_n) \in E\left(\prod_{i=1}^{n} G_i\right)$ si y sólo si existe $1 \leq j \leq n$ tal que $u_j v_j \in E(G_j)$ y $u_i = v_i$ para cada $i \neq j$. A partir de esta observación, puede inferirse la siguiente proposición.

Proposición 1.2.1. *Sean $\{G_1, \ldots, G_n\}$ una familia de gráficas, $I \subseteq \{1, \ldots, n\}$, $J = \{1, \ldots, n\} \setminus I$, y $\{g_i\}_{i \in I}$ un conjunto de vértices tales que $g_i \in V(G_i)$ para cada $i \in I$. Si $X = \{(v_1, \ldots, v_n) \in V(\prod_{i=1}^{n} G_i)\colon\ v_i = g_i \text{ si } i \in I, v_j \in G_j \text{ si } j \in J\}$, entonces*

$$\left(\prod_{i=1}^{n} G_i\right)[X] \cong \prod_{j \in J} G_j.$$

Como caso particular, si $I = \{i\}$,

$$\left(\prod_{i=1}^{n} G_i\right)[X] \cong \prod_{j \neq i} G_j.$$

La demostración se omite por ser sencilla pero bastante técnica, basta observar que el isomorfismo está dado por la proyección sobre el conjunto de coordenadas J.

Consideremos las siguientes tres familias de gráficas.

Definición 1.2.2. El ***hipercubo* n-*dimensional***, H_n $(n \geq 1)$, es la gráfica que tiene como conjunto de vértices al conjunto de todas las n-adas ordenadas de 0's y 1's, donde dos n-adas ordenadas son adyacentes si y sólo si difieren en exactamente una coordenada.

Definición 1.2.3. Definimos a la familia R_n recursivamente:

- $R_1 = K_2$,
- $R_{n+1} = R_n \square K_2$.

Definición 1.2.4. Los vértices de S_n son los elementos del conjunto potencia de A_n, donde $A_n = \{1, 2, 3, \ldots, n\}$; XY está en las aristas de S_n si y sólo si la diferencia simétrica de X con Y tiene exactamente un elemento, es decir, $|X \triangle Y| = 1$.

En esta sección demostraremos que estas tres familias de gráficas son isomorfas, por lo tanto, tendremos tres definiciones distintas para trabajar con los hipercubos n-dimensionales.

Lema 1.2.5. *En S_n, $|X \triangle Y| = 1$ si y sólo si existe un único $j \in \{1, 2, 3, \ldots, n\}$ tal que $\chi_j(X) \neq \chi_j(Y)$, donde $\chi_i : \mathcal{P}(A_n) \to \{0, 1\}$ tal que $\chi_i(Y) = \begin{cases} 1 & \text{si } i \in Y, \\ 0 & \text{si } i \notin Y. \end{cases}$*

Demostración. Recordemos que $X \triangle Y = (X \setminus Y) \cup (Y \setminus X)$, entonces $|X \triangle Y| = 1$ si y sólo si $|(X \setminus Y) \cup (Y \setminus X)| = 1$. Sabemos que $(X \setminus Y) \cap (Y \setminus X) = \varnothing$, por lo que $|(X \setminus Y) \cup (Y \setminus X)| = |(X \setminus Y)| + |(Y \setminus X)| = 1$. Sin pérdida de generalidad $|X \setminus Y| = 1$ y $|Y \setminus X| = 0$, entonces existe un único $z \in X \setminus Y$ y $Y \subseteq X$ por lo tanto $\chi_z(X) = 1$, $\chi_z(Y) = 0$ y $Y \subseteq X$, entonces $\chi_i(X) = \chi_i(Y)$ para todo $i \neq z$.

Si existe un único $1 \leq k \leq n$ tal que $\chi_k(X) \neq \chi_k(Y)$, supongamos sin pérdida de generalidad que $\chi_k(X) = 1$ y que $\chi_k(Y) = 0$ entonces $k \in X$ y $k \notin Y$ infiriendo que $k \in X \setminus Y$. Ahora vemos que $Y \setminus X = \varnothing$, de lo contrario vemos que existe k' tal que $k' \in Y$ y $k' \notin X$, claramente $k' \neq k$ y $\chi_{k'}(X) \neq \chi_{k'}(Y)$ llegando a una contradicción. Por tanto $Y \setminus X = \varnothing$ y por la unicidad de k se cumple que $|X \setminus Y| = 1$. Entonces $|(X \setminus Y) \cup (Y \setminus X)| = 1$ y por lo tanto $|X \triangle Y| = 1$. ■

Teorema 1.2.6. *Para cada entero $n \geq 1$ se cumple que $H_n \cong R_n \cong S_n$.*

Demostración. Veamos primero que $S_n \cong H_n$.

Como $A_n = \{1, 2, 3, \ldots, n\}$, para cada $i \in A_n$ definimos $\chi_i : \mathcal{P}(A_n) \to \{0, 1\}$ tal que

$$\chi_i(Y) = \begin{cases} 1 & \text{si } i \in Y, \\ 0 & \text{si } i \notin Y. \end{cases}$$

Sea $\varphi : \mathcal{P}(A_n) \to \{0, 1\}^n$, donde estamos denotando por $\{0, 1\}^n$ al producto cartesiano de n copias del conjunto $\{0, 1\}$, tal que $\varphi(Y) = (\chi_n(Y), \chi_{n-1}(Y), \ldots, \chi_1(Y))$, entonces $\varphi(Y) \in V(H_n)$ y resulta claro que φ está bien definida.

Para cada $i \in A_n$, definimos $\zeta_i : \{0,1\} \to \mathcal{P}(A_n)$ por

$$\zeta_i(x) = \begin{cases} \{i\} & \text{si } x = 1, \\ \varnothing & \text{si } x = 0; \end{cases}$$

si $y = (k_n, k_{n-1}, \ldots, k_1)$ entonces podemos definir a $\psi : \{0,1\}^n \to \mathcal{P}(A_n)$ de la siguiente forma

$$\psi(y) = \zeta_n(k_n) \cup \zeta_{n-1}(k_{n-1}) \cup \cdots \cup \zeta_1(k_1) = \bigcup_{i=1}^{n} \zeta_i(k_i).$$

Es fácil observar que

$$\zeta_i(\chi_i(Y)) = \begin{cases} \{i\} & \text{si } i \in Y, \\ \varnothing & \text{si } i \notin Y; \end{cases}$$

de esta manera se tiene

$$\begin{aligned} \psi \circ \varphi(Y) &= \psi(\chi_n(Y), \chi_{n-1}(Y), \ldots, \chi_1(Y)) \\ &= \zeta_n(\chi_n(Y)) \cup \zeta_{n-1}(\chi_{n-1}(Y)) \cup \cdots \cup \zeta_1(\chi_1(Y)) \\ &= Y, \end{aligned}$$

por lo tanto, $\psi \circ \varphi = 1_{\mathcal{P}(A_n)}$.

Por otro lado, si $y \in \{0,1\}^n$, entonces $k_j = 1$ si y sólo si $j \in \bigcup_{i=1}^{n} \zeta_i(k_i)$ si y sólo si $\chi_j(\bigcup_{i=1}^{n} \zeta_i(k_i)) = 1$. Además, $k_j = 0$ si y sólo si $j \notin \bigcup_{i=1}^{n} \zeta_i(k_i)$ si y sólo si $\chi_j(\bigcup_{i=1}^{n} \zeta_i(k_i)) = 0$. Por lo tanto

$$\begin{aligned} \varphi \circ \psi(y) &= \varphi(\zeta_n(k_n) \cup \zeta_{n-1}(k_{n-1}) \cup \cdots \cup \zeta_1(k_1)) \\ &= \left(\chi_n \left(\bigcup_{i=1}^{n} \zeta_i(k_n) \right), \chi_{n-1} \left(\bigcup_{i=1}^{n} \zeta_i(k_{n-1}) \right), \ldots, \chi_1 \left(\bigcup_{i=1}^{n} \zeta_i(k_1) \right) \right) \\ &= (k_n, k_{n-1}, \ldots, k_1). \end{aligned}$$

De esta manera podemos concluir que $\varphi \circ \psi = 1_{\{0,1\}^n}$. Así, obtenemos que φ es una función biyectiva. Para demostrar que φ es un isomorfismo, observemos que $XY \in E(S_n)$ si y sólo si $|X \triangle Y| = 1$. Por el Lema 1.2.5, ésto sucede si y sólo si existe

un único $j \in \{1, 2, 3, \ldots, n\}$ tal que $\chi_j(X) \neq \chi_j(Y)$ si y sólo si $\varphi(X)\varphi(Y) \in E(H_n)$; por lo tanto φ es un isomorfismo entre H_n y S_n.

Veamos ahora que $R_n \cong H_n$. Sabemos que $R_1 = K_2$, donde $V(R_1) = \{u, v\}$ y $E(R_1) = \{uv\}$. Sabemos también que $V(H_1) = \{(0), (1)\}$ y $E(H_1) = \{(0)(1)\}$. Podemos definir a $\varphi_1 : V(R_1) \to V(H_1)$ por

$$\varphi_1(u) = (0), \varphi_1(v) = (1).$$

Resulta claro que φ_1 es una función biyectiva con inversa $\varphi_1^{-1} : V(H_1) \to V(R_1)$ dada por $\varphi_1^{-1}((0)) = u, \varphi_1^{-1}((1)) = v$. Más aún, como $E(R_1) = \{uv\}$ y $E(H_1) = \{(0)(1)\}$, la biyección φ es un isomorfismo.

Entonces, para cada entero $n \geq 2$ podemos definir a $\varphi_n : V(R_n) \to V(H_n)$ como

$$\varphi_n(x_0, x_1, \ldots, x_n) = (\varphi_1(x_0), \varphi_1(x_1), \ldots, \varphi_1(x_n)).$$

Resulta claro que φ_n está bien definida y que tiene por inversa a la función $\varphi_n^{-1} : V(H_n) \to V(R_n)$ dada por

$$\varphi_n^{-1}(y_0, y_1, \ldots, y_n) = (\varphi_1^{-1}(y_0), \varphi_1^{-1}(y_1), \ldots, \varphi_1^{-1}(y_n)).$$

Por lo tanto φ_n es una función biyectiva, y basta entonces verificar que preserva las aristas. Demostraremos por inducción sobre n que $xy \in E(R_n)$ si y sólo si $\varphi_1(x)\varphi_1(y) \in E(H_n)$.

Para $n = 1$ ya lo hemos verificado. Ahora supongamos que φ_{n-1} es un isomorfismo entre R_{n-1} y H_{n-1} y sean $x = (x_0, x_1, \ldots, x_n)$ y $z = (z_0, z_1, \ldots, z_n)$. Sabemos que $R_n = R_{n-1} \square K_2$ por lo tanto $V(R_n) = X_1 \cup Y_1$, donde

$$X_1 = \{(x_0, x_1, \ldots, x_{n-1}, u) \mid x_i \in \{u, v\}, 1 \leq i \leq n-1\},$$

$$Y_1 = \{(x_0, x_1, \ldots, x_{n-1}, v) \mid x_i \in \{u, v\}, 1 \leq i \leq n-1\}.$$

Claramente $X_1 \cap Y_1 = \varnothing$; además la Proposición 1.2.1 garantiza que $R_n[X_1] \cong R_{n-1}$ y $R_n[Y_1] \cong R_{n-1}$.

Análogamente puede observarse que si

$$X_2 = \{(y_0, y_1, \ldots, y_{n-1}, 0) \mid y_i \in \{0, 1\}, 1 \leq i \leq n-1\},$$

$$Y_2 = \{(y_0, y_1, \ldots, y_{n-1}, 1) \mid y_i \in \{0,1\}, 1 \leq i \leq n-1\},$$

entonces $X_2 \cap Y_2 = \varnothing$ y $X_2 \cup Y_2 = V(H_n)$. Además, como todos los vértices en X_2 tienen la misma coordenada final, no es difícil obtener que $H_n[X_2] \cong H_{n-1}$ y, análogamente, $H_n[Y_2] \cong H_{n-1}$.

Por las definiciones de φ_{n-1} y φ_n, resulta claro que

$$\varphi_n(x_0, x_1, \ldots, x_n) = (\varphi_{n-1}(x_0, x_1, \ldots, x_{n-1}), \varphi_1(x_n)).$$

Podemos considerar dos casos. Si $x_n = z_n$, podemos suponer sin pérdida de generalidad que $x_n = z_n = u$. Entonces $z, x \in X_1$ y por lo tanto $xz \in E(R_n[X_1])$. Si definimos a $\pi : R_n[X_1] \to R_{n-1}$ por $\pi((x_0, \ldots, x_{n-1}, u)) = (x_0, \ldots, x_{n-1})$ y a $\eta : H_{n-1} \to H[X_2]$ por $\eta((w_0, \ldots, w_{n-1})) = (w_0, \ldots, w_{n-1}, 0)$, resulta claro que éstos son los isomorfismos antes mencionados, por lo tanto, al definir $\psi = \eta \circ \varphi_{n-1} \circ \pi$, y recordando que la composición de isomorfismos es un isomorfismo, tenemos que ψ es un isomorfismo. Pero $\psi(x_0, \ldots, x_{n-1}, u) = (\varphi_1(x_0), \ldots, \varphi_1(x_{n-1}), 0) = \varphi_n(x_0, \ldots, x_n)$. Así, $\psi = \varphi_n\big|_{R_n[X_1]}$, por lo que el diagrama de la Figura 1.7 es conmutativo y $\varphi_n\big|_{R_n[X_1]} : R_n[X_1] \to H[X_2]$ es un isormorfismo. En particular, $xz \in E(R_n[X_1])$ si y sólo si $\varphi_n(x)\varphi_n(z) \in E(H_n[X_2])$ y por lo tanto $xz \in E(R_n)$ si y sólo si $\varphi(x)\varphi(z) \in E(H_n)$.

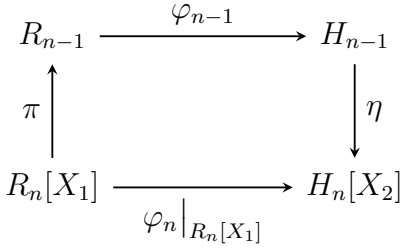


Figura 1.7: $\varphi_n\big|_{R_n[X_1]} = \psi = \eta \circ \varphi_{n-1} \circ \pi$

Ahora supongamos $x_n \neq z_n$. Inferimos que $xz \in E(R_n)$ si y sólo si $x_i = z_i$, $1 \leq i \leq n-1$, si y sólo si, sin pérdida de generalidad $x = (x_0, x_1, \ldots, x_{n-1}, u)$ y $z = (x_0, x_1, \ldots, x_{n-1}, v)$ si y sólo si $\varphi(x) = (\varphi_1(x_0), \varphi_1(x_1), \ldots, \varphi_1(x_{n-1}), 0)$ y $\varphi(z) = (\varphi_1(x_0), \varphi_1(x_1), \ldots, \varphi_1(x_{n-1}), 1)$ si y sólo si $\varphi(x)\varphi(z) \in E(H_n)$ y $\varphi_1(x_n) \neq \varphi_1(z_n)$. Por lo tanto $R_n \cong H_n$ y por lo tanto $S_n \cong H_n \cong R_n$. ■

En el hipercubo d-dimensional H_d con $n = 2^d$ vértices, para cada vértice x, hay una etiqueta distinta asociada con cada una de las aristas incidentes; la etiqueta entre el vértice x y el vértice y, es la posición de la entrada en la que los vértices x y y difieren, llamada dimensión. Dado un vértice x del hipercubo H_d, denotaremos por $E(x)$ al conjunto formado por todas las aristas incidentes a x. Sea $\lambda_x : E(x) \to \{1, 2, \ldots, d\}$ una función inyectiva que define la etiqueta de la arista para el vértice x, es decir, $\lambda_x(xy)$ indica la entrada en la que x difiere de y.

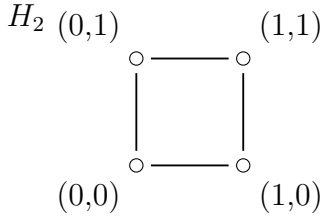


Figura 1.8: Hipercubo 2-dimensional

En la Figura 1.8 podemos observar que:

$$\begin{aligned}
&\lambda_{(0,0)}((0,0)(0,1)) = 1, && \lambda_{(1,0)}((1,0)(1,1)) = 1,\\
&\lambda_{(0,0)}((0,0)(1,0)) = 2, && \lambda_{(1,0)}((1,0)(0,0)) = 2,\\
&\lambda_{(0,1)}((0,1)(0,0)) = 1, && \lambda_{(1,1)}((1,1)(1,0)) = 1,\\
&\lambda_{(0,1)}((0,1)(1,1)) = 2, && \lambda_{(1,1)}((1,1)(0,1)) = 2.
\end{aligned}$$

1.3. Notación de Landau

La notación de Landau [10] la utilizamos en algoritmos cuando el argumento de una función tiende a infinito y nos sirve para dar una cota superior asintótica, una cota inferior asintótica o una cota ajustada asintótica, es decir, una cota ya sea superior, inferior o un intervalo. La aplicamos para describir la complejidad de un algoritmo, esta notación nos mostrará el peor escenario que podemos obtener al realizar un algoritmo, al decir peor escenario nos referimos a la cota más grande del tiempo de ejecución requerido o del espacio que a lo más se necesita en memoria.

Cota Superior Asintótica $O(g(x))$, conocida también como notación de las grandes O's. Una función $f(x)$ pertenece a $O(g(x))$ siempre que exista una constante positiva

c tal que a partir de un valor x_0, $f(x)$ no sobrepasa a $cg(x)$, lo que implica que $f(x)$ es inferior a $g(x)$ a partir de un valor dado, salvo por un factor constante c.

Cota Inferior Asintótica $\Omega(g(x))$. Una función $f(x)$ pertenece a $\Omega(g(x))$ siempre que exista una constante positiva c tal que a partir de un valor x_0, $cg(x)$ no supera a $f(x)$, lo que implica que $f(x)$ es superior a $g(x)$ a partir de un valor dado, salvo por un factor constante c.

Cota Ajustada Asintótica $\Theta(g(x))$, esta cota nos sirve tanto de cota superior como de cota inferior de una función cuando el argumento tiende a infinito. Una función $f(x)$ pertenece a $\Theta(g(x))$ siempre que existan dos constantes positivas c_1 y c_2 tales que a partir de un valor x_0, $f(x)$ se encuentre atrapada entre $c_1g(x)$ y $c_2g(x)$, lo que nos indica que nuestras funciones $f(x)$ y $g(x)$ son iguales a partir de un valor dado, excepto por una constante c_1 y c_2 respectivamente.

En este trabajo utilizaremos esta notación al hablar de la complejidad de nuestros algoritmos.

1.4. Coeficientes Binomiales

A continuación presentamos algunas propiedades relativas a coeficientes binomiales, de los cuales se hará uso durante el estudio del comportamiento de los distintos algoritmos.

Proposición 1.4.1. *Para cualesquiera $r, n \in \mathbb{N}$, con $1 \leq r \leq n$, se cumple que*

$$\binom{n}{r} = \binom{n}{n-r}.$$

Demostración. De la definición de $\binom{n}{r}$ se sigue que $\binom{n}{r} = \frac{n!}{(n-r)!r!} = \binom{n}{n-r}$. ■

Proposición 1.4.2. *Para cualesquiera $r, n \in \mathbb{N}$, con $1 \leq r \leq n$, se cumple que*

$$\binom{n}{r} = \binom{n-1}{r-1} + \binom{n-1}{r}.$$

Demostración. Notemos que

$$\begin{aligned}\binom{n-1}{r-1}+\binom{n-1}{r} &= \frac{(n-1)!}{(n-r)!(r-1)!}+\frac{(n-1)!}{(n-r-1)!(r)!}\\ &= \frac{(n-1)!(r+n-r)}{(n-r)!(r)!}\\ &= \frac{n!}{(n-r)!(r)!}\\ &= \binom{n}{r}.\end{aligned}$$

■

Proposición 1.4.3. *Para cualesquiera $r, n \in \mathbb{N}$, con $\lfloor n/2 \rfloor \leq r \leq \lceil n/2 \rceil$ se cumple*

$$\max_{1\leq s\leq n}\left\{\binom{n}{s}\right\} = \binom{n}{r}.$$

Demostración. Notemos que

$$\binom{n}{r-1} = \frac{n!}{(r-1)!(n-r+1)!} = \frac{n!}{r!(n-r)!}\ \frac{r}{n-r+1} = \binom{n}{r}\frac{k}{n-r+1},$$

de lo cual se sigue que $\binom{n}{r-1} \leq \binom{n}{r}$ si y sólo si $\frac{r}{n-r+1} \leq 1$. Lo cual es equivalente a

$$r \leq n-r+1 \iff 2r \leq n+1 \iff r \leq \frac{n+1}{2} \iff r \leq \left\lceil \frac{n}{2} \right\rceil.$$

A partir de lo anterior y la simetría de los coeficientes binomiales obtenida en la Proposición 1.4.1, obtenemos que $\binom{n}{s} \leq \binom{n}{r}$ para $\lfloor n/2 \rfloor \leq r \leq \lceil n/2 \rceil$, de lo cual se sigue el resultado. ■

Proposición 1.4.4. *Para cualesquiera $r, n \in \mathbb{N}$, con $\lfloor n/2 \rfloor \leq r \leq \lceil n/2 \rceil$ se cumple*

$$\max_{1\leq s\leq n}\left\{\binom{n}{s-1}+\binom{n}{s}+\binom{n}{s+1}\right\} = \binom{n}{r-1}+\binom{n}{r}+\binom{n}{r+1}.$$

Demostración. En la prueba de la proposición anterior se muestra el comportamiento monótono de los coeficientes binomiales, es decir, $\binom{n}{r-1} \leq \binom{n}{r}$ siempre que $0 \leq r \leq \left\lceil \frac{n}{2} \right\rceil$. La Proposición 1.4.1 y la monotonía de los coeficientes binomiales implican directamente el resultado. ■

Capítulo 2

Descontaminación por Agentes Móviles

2.1. Introducción

En el modelo sobre el que trabajaremos, conocido como modelo local, un grupo de agentes será situado en una base, o vértice fuente, y cada agente es libre de moverse de la base a cualquier vértice vecino a través de la arista que los conecta. En cualquier momento del tiempo cada vértice de la red puede encontrarse en cualquiera de los tres estados siguientes: resguardado, limpio (o desinfectado) y contaminado. Cabe destacar que bajo las estrategias que analizaremos no se permite la recontaminación. Inicialmente todos los vértices, excepto el vértice fuente, se encuentran contaminados.

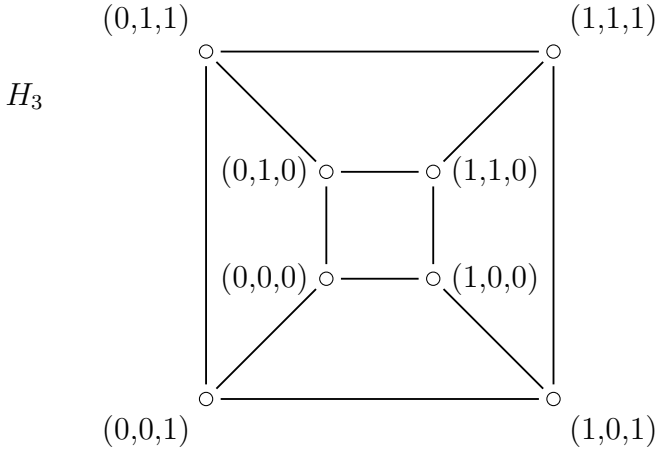
1. Resguardado: un vértice se encuentra resguardado cuando contiene al menos un agente.

2. Limpio: cuando un agente ha estado en el vértice y todos sus vértices vecinos se encuentran limpios o resguardados.

3. Contaminado: en cualquier otro caso.

Un grupo de agentes móviles autónomos operan en G, cada agente es asociado con un identificador distinto, cada agente puede realizar un cálculo local, puede moverse de un vértice a otro vértice vecino y tiene una cierta memoria local ($O(\log n)$ bits son suficiente para todos los algoritmos). Además, cada agente obedece el mismo conjunto de reglas de comportamiento, sabe que está operando en un hipercubo y se puede comunicar con otros agentes, esto sólo cuando se encuentran los dos al mismo tiempo en el mismo vértice. El entorno se dice asincrónico, es decir, cada acción que el agente ejecuta, (como un cálculo o movimiento), toma un lapso finito de tiempo, pero este lapso es impredecible.

Vamos a considerar al hipercubo H_d organizado en $d+1$ niveles: el nivel $i \in \{0, 1, \ldots, d\}$ consiste de todos los vértices cuya representación binaria contenga exactamente i unos. Nótese que, para el hipercubo H_d, el nivel 0 sólo contiene al vértice $(0, 0, \ldots, 0)$ y el nivel d sólo contiene al vértice $(1, 1, \ldots, 1)$. Por ejemplo, para el hipercubo 3-dimensional H_3 tenemos que $(0, 0, 0)$ está en el nivel 0, los vértices $(1, 0, 0), (0, 1, 0)$ y $(0, 0, 1)$ pertenecen al nivel 1, los vértices $(1, 1, 0), (1, 0, 1)$ y $(0, 1, 1)$ conforman el nivel 2, y $(1, 1, 1)$ está contenido en el nivel 3. Claramente todos los vértices en el nivel i están conectados únicamente a los vértices del nivel $i-1$ y a los del nivel $i+1$. Podemos observar en la Figura 2.1 que el vértice $(1, 0, 0)$ que es del nivel 1, es adyacente a los vértices $(0, 0, 0)$, del nivel 0, $(1, 0, 1)$ y $(1, 1, 0)$ que son del nivel 2. La "coordenada más significativa" del vértice x es la coordenada en la que se encuentra el último uno, de derecha a izquierda, en la n-ada, y su posición se denotará por $m(x)$, es decir, si $x = (a_d, \ldots, a_2, a_1)$, entonces $m(x) = \text{máx}\{k \mid a_k = 1, 1 \leq k \leq d\}$.



$m((0,0,0)) = 0$
$m((0,0,1)) = 1$
$m((0,1,0)) = 2$
$m((1,0,0)) = 3$
$m((0,1,1)) = 2$
$m((1,0,1)) = 3$
$m((1,1,0)) = 3$
$m((1,1,1)) = 3$

Figura 2.1: El hipercubo H_3 y la coordenada más significativa para cada vértice.

Sea T_d un árbol generador de amplitud primaria de H_d, con raíz en la fuente (el vértice $(0, 0, \ldots, 0, 0)$), que se define de la siguiente forma: Hay una arista en el árbol generador entre el vértice x y todos los vértices en el siguiente nivel cuya representación binaria difiere en una posición mayor que $m(x)$. Este árbol generador también es llamado árbol de transmisión ya que se utiliza para ejecutar una transmisión óptima en el hipercubo: Un vértice x del nivel i que recibe un mensaje desde un nivel anterior lo reenviará a todos los vértices en el nivel $j > i$ a los que esté conectado mediante una trayectoria en dicho árbol.

A partir de la definición del árbol generador de transmisión T_d podemos observar cierto comportamiento en sus vértices, por lo cual diremos que un vértice es del tipo $T(k)$ si tiene k hijos. Notemos que el árbol generador T_d cumple lo siguiente:

- Un vértice del tipo $T(0)$ es una hoja.
- Un vértice del tipo $T(1)$ es un vértice con un hijo.
- Un vértice del tipo $T(k)$ es un vértice con k hijos del tipo $T(0), \ldots, T(k-1)$.
- El vértice fuente $(0, 0, \ldots, 0)$ de T_d es el único del tipo $T(d)$.

Sean x y y dos vértices vecinos ($(x, y) \in E(x)$). El vértice y es llamado un vecino pequeño del vértice x si $\lambda_x(x, y) \leq m(x)$ y es llamado un vecino grande de x si $\lambda_x(x, y) > m(x)$. Notemos que los vecinos grandes de x son los hijos de x en este árbol de transmisión.

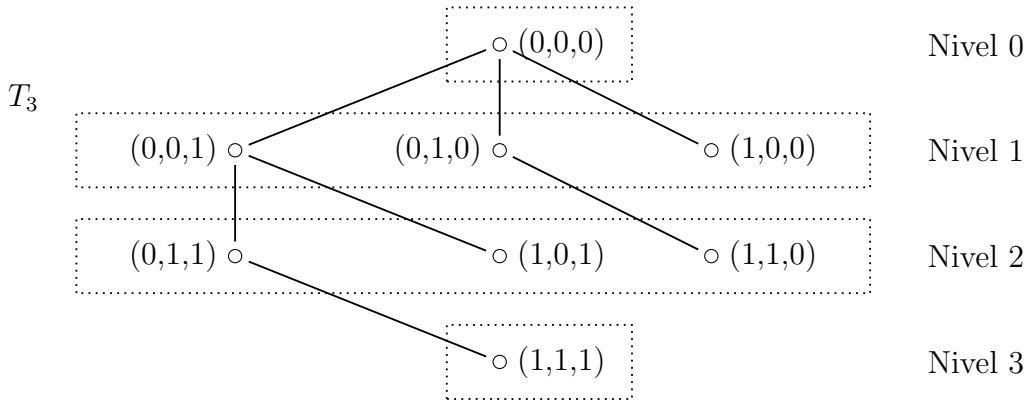


Figura 2.2: Ejemplo de un árbol de transmisión T_3 en H_3.

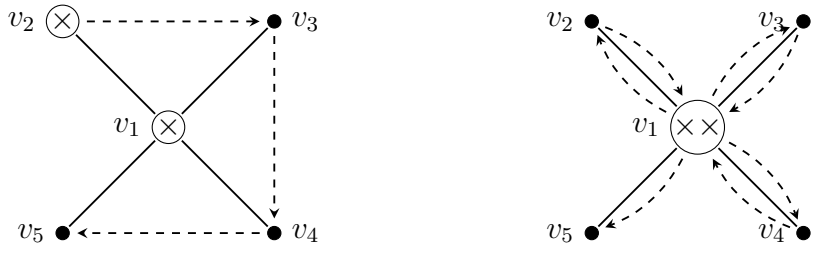
En la Figura 2.2 vemos que el vértice $(0,0,0)$ es del tipo $T(3)$ el cual tiene 3 hijos, uno del tipo $T(0)$, el vértice $(1,0,0)$, uno del tipo $T(1)$, el $(0,1,0)$, que tiene un hijo, el vértice $(1,1,0)$, y uno del tipo $T(2)$, el $(0,0,1)$, que tiene dos hijos, los vértices $(0,1,1)$ y $(1,0,1)$. Además podemos observar que

$\lambda_{(0,0,1)}((0,0,1)(0,0,0)) = 1 \leq m(0,0,1) = 1$ donde $(0,0,0)$ es vecino pequeño,
$\lambda_{(0,0,1)}((0,0,1)(0,1,1)) = 2 > m(0,0,1) = 1$ donde $(0,1,1)$ es vecino grande,
$\lambda_{(0,0,1)}((0,0,1)(1,0,1)) = 3 > m(0,0,1) = 1$ donde $(1,0,1)$ es vecino grande.

2.2. Modelo Clásico

Ahora vamos a dar una introducción del modelo clásico, también llamado búsqueda por vértices. En este modelo los agentes no se van a mover a lo largo de las aristas, si no que se van a situar en los vértices como guardias y van a realizar saltos. En este modelo tampoco hay una base establecida al inicio, sino que vamos a colocar a los agentes donde lo consideremos conveniente, por lo que la desinfección se comienza en un vértice elegido.

A partir de lo anterior, observamos que el modelo clásico requiere un menor o igual número de movimientos que el modelo local, debido a que los movimientos de los agentes no están restringidos por las aristas de la gráfica. En la Figura 2.3 se muestra una gráfica cuya limpieza requiere tres movimientos bajo el modelo clásico, mientras que bajo el modelo local requiere siete, aunque en ambos se requieren únicamente dos agentes.



Modelo clásico Modelo local

Figura 2.3: Descontaminación de una gráfica.

Debemos tener cuidado al mover los agentes para evitar una recontaminación, la cual puede ocurrir si después de haber sido limpiado por un agente, un vértice se queda vacío y alguno de sus vecinos está contaminado. La búsqueda se termina cuando ya no hay vértices contaminados.

El número mínimo de agentes que se necesitan para limpiar todos los vértices de G se llama ***número de búsqueda en vértices*** de G, $ns(G)$. A continuación estableceremos un cota inferior sobre el número de agentes necesarios para limpiar el hipercubo; esta cota es válida para todos los modelos con los que trabajemos. La cota inferior en el número de agentes está ligada al concepto de amplitud de trayectoria, un parámetro clásico en la Teoría de Gráficas. Una ***descomposición en trayectorias*** de una gráfica $G = (V, E)$ es una colección $\{X_1, X_2, \ldots, X_r\}$ de subconjuntos de V donde

- $\bigcup_{i=1}^{r} X_i = V$.
- Para toda $xy \in E$, existe $i \in \{1, \ldots, r\}$ tal que $x, y \in X_i$.
- Para cualesquiera i, j, k con $1 \leq i < j < k \leq r$, $X_i \cap X_k \subseteq X_j$.

La ***amplitud*** de una descomposición en trayectorias $(X_1, X_2, \ldots, X_r)$ de G la definimos como el $\max_{1 \leq i \leq r} |X_i| - 1$, y la ***amplitud de trayectoria*** (*pathwidth*) de G es la amplitud mínima sobre sus descomposiciones.

En la Figura 2.4 se muestra una gráfica que tiene dos sencillos ejemplos de descomposiciones de trayectorias. Para el primero, consideremos los conjuntos

$$X_1 = \{v_1, v_2, v_3\}, \quad X_2 = \{v_3, v_4, v_5, v_6\} \text{ y } \quad X_3 = \{v_3, v_7, v_8, v_9\},$$

donde se cumple que $X_1 \cap X_3 = \{v_3\} \subset X_2$.

Por otro lado, podemos considerar ahora a los conjuntos:

$$Y_1 = \{v_1, v_2, v_3\},\ Y_2 = \{v_3, v_4, v_5\},\ Y_3 = \{v_3, v_5, v_6\},\ Y_4 = \{v_3, v_7\} \text{ y } Y_5 = \{v_8, v_7, v_9\},$$

en este caso podemos verificar que para cualquier terna Y_i, Y_j, Y_k con $1 \leq i < j < k \leq 5$ se cumple que $Y_i \cap Y_k = \varnothing$, o bien, $Y_i \cap Y_k = \{v_3\}$, y que $\{v_3\} \subseteq Y_j$, donde $j \in \{2, 3, 4\}$. Entonces, podemos observar que la descomposición (X_1, X_2, X_3) tiene amplitud 3 y la descomposición $(Y_1, Y_2, Y_3, Y_4, Y_5)$ tiene amplitud 2.

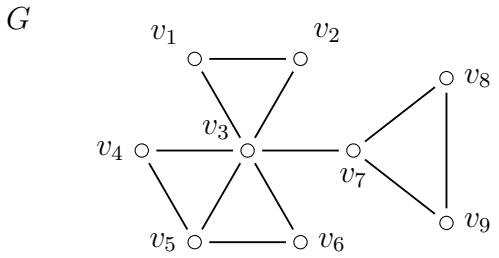


Figura 2.4: Ejemplo de una descomposicón de trayectorias.

Los siguientes dos teoremas fueron demostrados en [11] y en [4] respectivamente.

Teorema 2.2.1. *[11] Para toda gráfica G,* $ns(G) = pathwidth(G) + 1$. ■

Teorema 2.2.2. *[4] Si* H_d *es un hipercubo, entonces* $pathwidth(H_d) = \Theta(\frac{n}{\sqrt{\log n}})$. ■

Hemos omitido la demostración de los dos teoremas anteriores debido a su longitud y a sus características técnicas. Dichos resultados nos servirán para establecer lo siguiente.

Teorema 2.2.3. *Para resolver el problema de búsqueda conexa monótona, se necesitan* $\Omega(\frac{n}{\sqrt{logn}})$ *agentes.*

Demostración. Por el Teorema 2.2.2 sabemos que la amplitud de trayectoria de un hipercubo es $\Theta(\frac{n}{\sqrt{\log n}})$. Observemos que en el modelo local no hay saltos por lo que puede llevarnos más pasos o tiempo que el modelo clásico, ya que en este hay saltos, entonces la cota inferior para el modelo clásico descrita en el Teorema 2.2.1 es la misma que en el modelo local. ■

Capítulo 3

Estrategia de Limpieza con Coordinador

3.1. Modelo Local

Asumiremos que el hipercubo H_d en el que trabajemos es de grado par y además $d \geq 4$. La primera estrategia que presentaremos es óptima con respecto al número de agentes. La idea principal es que todos los agentes comiencen en la misma base y que sus movimientos sean coordinados por un agente lider. Los agentes visitan todos los vértices mientras protegen el sistema de una recontaminación.

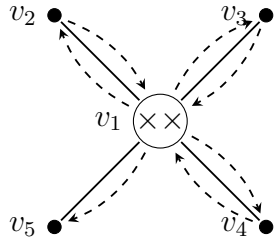


Figura 3.1: Descontaminación modelo local.

Recordemos que descontaminando bajo el modelo local requeriremos emplear un mayor o igual número de movimientos que empleando el modelo clásico.

3.1.1. Propiedades Básicas de los Árboles Generadores

Propiedad 3.1.1. *Sea T_d un árbol generador para el hipercubo:*

(a) *El número de vértices en el nivel l es $\binom{d}{l}$.*

(b) *El nivel 0 no tiene hojas, los niveles $l > 0$ tienen $\binom{d-1}{l-1}$ hojas, y el número total de hojas es 2^{d-1}.*

(c) *En el nivel 0 hay un único vértice de tipo $T(d)$. En los niveles $l > 0$ hay $\binom{d-k-l}{l-1}$ vértices del tipo $T(k)$ con $0 \leq k \leq d-l$.*

Demostración.

(a) Notemos que cada vértice en T_d tiene d entradas en su coordenada en representación binaria. Para estar en el nivel l deben haber l 1's exactamente en la coordenada. Por lo que $\binom{d}{l}$ son las formas de acomodar l 1's en d entradas.

(b) Procederemos por inducción sobre el número de hojas. El nivel 0 no tiene hojas, ya que es del tipo $T(d)$. El nivel 1 tiene una hoja pues el nivel 0 es del tipo $T(d)$. Supongamos que el nivel l tiene $\binom{d-1}{l-1}$ hojas. Observemos que en el nivel l el número de vértices menos el número de hojas es el número de vértices del tipo $T(k)$, con $k \neq 0$, donde cada uno de los vértices tiene exactamente un hijo en el nivel $l+1$ del tipo $T(0)$, es decir, una hoja. Por tanto hay $\binom{d}{l} - \binom{d-1}{l-1} = \binom{d-1}{l}$ hojas en el nivel $l+1$. El número total de hojas es

$$\sum_{k=0}^{d-1} \binom{d-1}{k} = \binom{d-1}{0} + \binom{d-1}{1} + \cdots + \binom{d-1}{d-1} = 2^{d-1}.$$

(c) En el nivel 0 hay un único vértice del tipo $T(d)$, esto se da por construcción. Todo vértice del nivel 1 es hijo del único vértice del nivel 0, entonces hay un único vértice en el nivel 1 del tipo $T(k)$ con $k \in \{0, 1, 2, \ldots, d-1\}$, es decir hay $\binom{d-k-1}{1-1} = \binom{d-k-1}{0}$ vértices del tipo $T(k)$ en el nivel 1. Supongamos que en el nivel l hay $\binom{d-k-1}{l-1}$ vértices del tipo $T(k)$ para todo $0 \leq k \leq d-l$, entonces

queremos demostrar que en el nivel $l+1$ hay $\binom{d-j-1}{l}$ vértices del tipo $T(j)$ con $0 \leq j \leq d-l-1$. Demostración por inducción sobre j. Si $j=0$ entonces $\binom{d}{l} - \sum_{k=0}^{j=0} \binom{d-k-1}{l-1} = \binom{d}{l} - \binom{d-1}{l-1} = \binom{d-1}{l}$. Supongamos que para $j=m$ se cumple que $\binom{d}{l} - \sum_{k=0}^{m} \binom{d-k-1}{l-1} = \binom{d-m-1}{l}$. Entonces, si $j=m+1$, tenemos que

$$\begin{aligned}
\binom{d}{l} - \sum_{k=0}^{m+1} \binom{d-k-1}{l-1} &= \binom{d-m-1}{l} - \binom{d-(m+1)-1}{l-1} \\
&= \binom{d-m-1}{l} - \binom{d-m-2}{l-1} \\
&= \binom{d-m-2}{l}.
\end{aligned}$$

■

Para los vértices del hipercubo también podemos utilizar la siguiente etiquetación

$$\hat{x} = \begin{cases} \varnothing & \text{si } x \text{ es la fuente (o vértice fuente),} \\ \langle i_1, i_2, \ldots, i_l \rangle & \text{donde } i_k \text{ para } 1 \leq k \leq l \leq d, \text{ son las posiciones de} \\ & \text{de las entradas iguales a uno en las coordenadas} \\ & \text{binarias (de derecha a izquierda).} \end{cases}$$

Cuando utilizemos esta etiqueta consideraremos que los vértices del árbol generador en cada nivel están ordenados lexicográficamente y no de acuerdo a su representación binaria; utilizaremos los símbolos $<_R$ y $>_R$ al hacer comparaciones lexicográficas. Como ejemplo podemos considerar a los vértices $(0,0,1,1,0,1)$ y $(0,1,1,1,0,0,0)$ en el hipercubo 6 dimensional H_6, a los cuales corresponden las etiquetas $\langle 1,3,4 \rangle$ y $\langle 4,5 \rangle$ respectivamente, donde se cumple que $\langle 1,3,4 \rangle <_R \langle 4,5 \rangle$.

3.1.2. Estrategia de Limpieza

Uno de los agentes va a actuar como coordinador de todo el proceso de limpieza. La estrategia de limpieza se lleva a cabo en el árbol generador de transmisión, la estructura del árbol generador nos garantiza que cuando un nivel l esté completamente resguardado todos los agentes en el vértice $\hat{i}$ del nivel l pueden moverse al siguiente

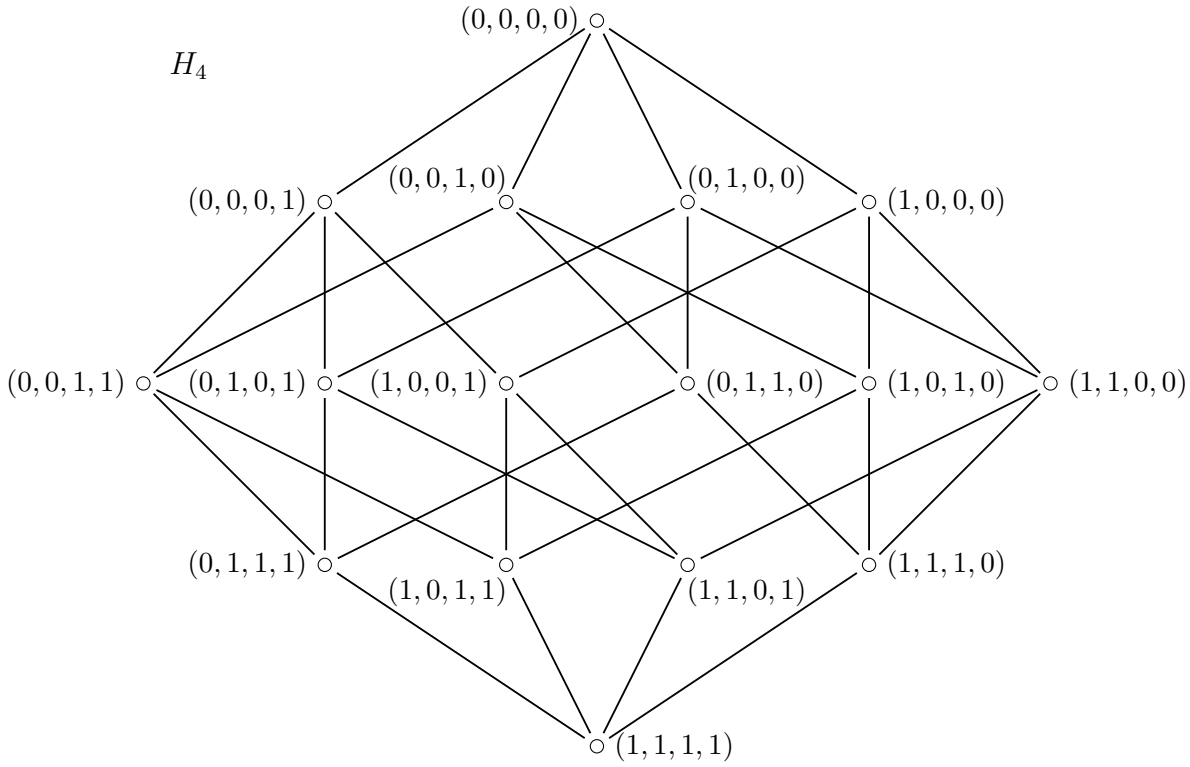


Figura 3.2: Ejemplo de un hipercubo 4 dimensional.

nivel sin incurrir en una recontaminación del vértice $\hat{i}$ si los agentes en los vértices $\hat{j} <_R \hat{i}$ (en el nivel l) ya se han movido al siguiente nivel. En otras palabras, el árbol generador y la etiquetación $\hat{x}$ definen una correcta estrategia de limpieza en orden para los vértices.

La idea principal es colocar los agentes suficientes en el vértice fuente $(0, 0, \ldots, 0)$ y coordinar sus movimientos en las aristas del árbol generador nivel por nivel. El grupo de agentes disponibles en la raíz es llamado ***conjunto de agentes disponibles***.

3.1.3. Algoritmo de Limpieza con Localidad

1. De la raíz al nivel uno.

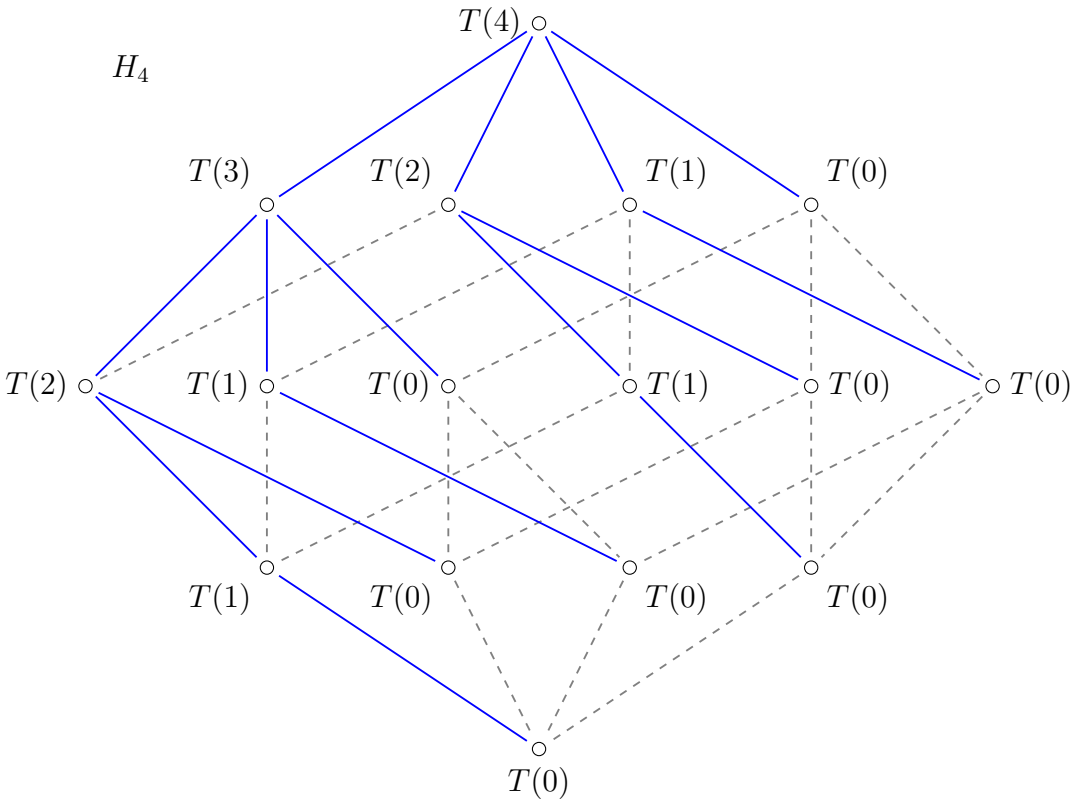


Figura 3.3: Las aristas verdes nos muestran el árbol generador del hipercubo 4 dimensional.

1.1 El coordinador guía a un agente distinto de la raíz a cada uno de sus d hijos, de los tipos $T(d-1), T(d-2), \ldots, T(0)$, y cada vez regresa a la raíz, esto lo lleva a cabo el coordinador d veces, lo que le lleva $2d$ unidades de tiempo.

2. Del nivel $l \geq 1$ al nivel $l+1 \leq d$ (el nivel l tiene un agente por vértice).

2.1 Antes de comenzar a limpiar los vértices en el nivel $l+1$ el coordinador regresa a la raíz para tomar el número necesario de agentes para limpiar el nivel $l+1$, (a saber $k-1$ agentes por vértice del tipo $T(k)$ con $0 \leq k \leq d-1$, excepto para los vértices del tipo $T(0)$ y $T(1)$, los cuales no requieren agentes extras). El coordinador envía $k-1$ agentes adicionales, sin un orden en específico a cada vértice del tipo $T(k)$, $k > 1$, al nivel l y luego

se mueve al primer vértice del nivel l.

2.2 Cuando k agentes están en un vértice del tipo $T(k)$ en el nivel l, estos son enviados por el árbol generador a sus hijos en el nivel $l+1$, guiados por el coordinador. Sean $\hat{n}_1, \hat{n}_2, \ldots, \hat{n}_m$, $m = \binom{d}{l}$, los vértices en el nivel l ordenados lexicográficamente. El coordinador escoge sucesivamente a cada vértice del nivel l siguiendo el orden lexicográfico y vértice a vértice, guiando a un agente por una arista del árbol generador al nivel $l+1$.

2.3 Cuando el coordinador alcanza una hoja del nivel l el agente al que iba guiando se convierte en disponible y regresa a la raíz. Hay que notar que cuando el coordinador alcanza el último vértice del nivel l, los únicos agentes activos son aquellos que están resguardando el nivel $l+1$.

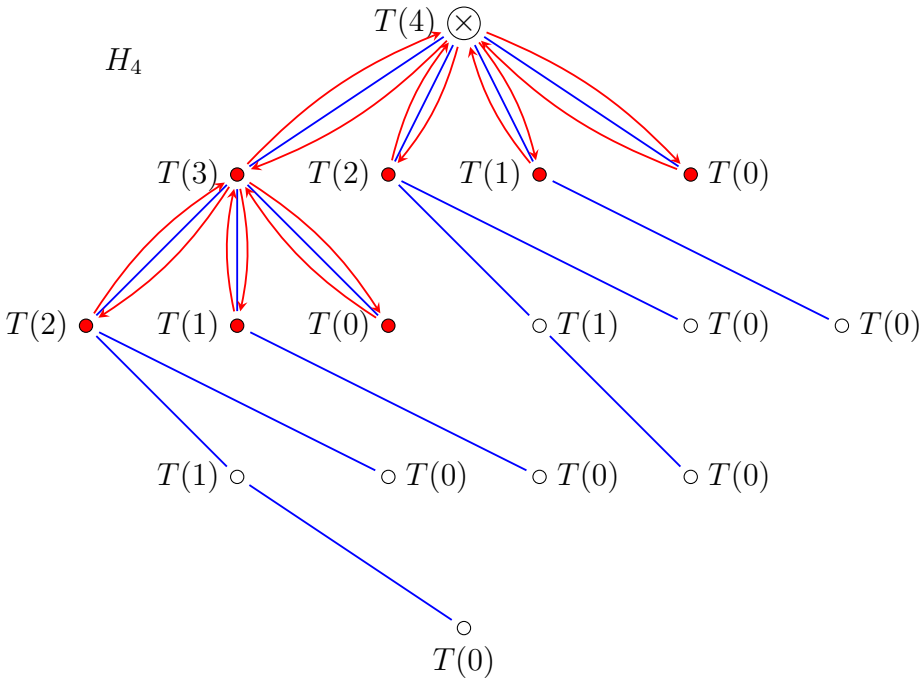


Figura 3.4: Estrategia de limpieza con localidad del árbol generador del hipercubo 4 dimensional.

Corrección y Complejidad

Corrección: Vamos a probar que el algoritmo de limpieza es correcto, esto es, todos los vértices del hipercubo van a ser desinfectados, y una vez que un vértice se

limpie, éste no podrá recontaminarse.

Sea x un vértice del nivel l, sea $N(x)$ el conjunto formado por los vecinos de x en H_d que pertenecen al nivel $l+1$ en T_d y sea $NT(x)$ el conjunto de los hijos de x en el nivel $l+1$ en el árbol generador de transmisión T_d. Notemos que $NT(x) \subset N(x)$ y además $N(x)$ puede contener también otros vecinos del nivel $l+1$.

Lema 3.1.2. *Si $z \in N(y) \setminus NT(y)$ entonces $z \in NT(x)$ para algún x tal que $\hat{x} <_R \hat{y}$*

Demostración. Si $z \in N(y) \setminus NT(y)$ esto quiere decir que z es vecino de y en el hipercubo, pero no en el árbol de transmisión. Por la forma en la que está definido el árbol de transmisión esto es posible únicamente si existe un vértice x tal que $\hat{x} <_R \hat{y}$ y tal que z es adyacente a x en el hipercubo. Consideremos al menor vértice (lexicográficamente) x en el mismo nivel que y con esta propiedad. Nuevamente, por la definición del árbol de transmisión, es claro que $z \in NT(x)$. ■

Lema 3.1.3. *En el algoritmo de limpieza cuando los agentes dejan sin protección un vértice x en un nivel l, todos los vecinos de x estan limpios o resguardados.*

Demostración. Esto es claramente cierto para la fuente, que es la base de los agentes. Asumamos que es cierto para todos los vértices en el nivel $0 \leq j < i$ y consideremos el nivel i. Los vértices en el nivel i están sólo conectados a vértices en los niveles $i-1$ e $i+1$. Cuando los vértices en el nivel i están enviando agentes al nivel $i+1$, por hipótesis inductiva y la estrategia de limpieza todos los vértices en el nivel $i-1$ están limpios. Ahora asumamos que la limpieza ocurre en un vértice y en el nivel i. Para $z \in N(y) \setminus NT(y)$, por el lema inmediato anterior, existe x tal que $z \in NT(x)$ y $\hat{x} <_R \hat{y}$. Por la estrategia de limpieza el coordinador visita todos los vértices en orden lexicográfico en cada nivel. Así que antes de que el coordinador llegue al vértice y, los agentes en el vértice x ya han sido enviados al nivel $i+1$ y todos los vértices $z \in N(y) \setminus NT(y)$ ya han sido resguardados por un agente. Si y es un vértice del tipo $T(j)$, $j \geq 1$, por el paso 2.1. del algoritmo de limpieza, los agentes suficientes son enviados del conjunto de agentes disponibles en la base al vértice y a limpiar los hijos de y en el árbol generador de transmisión. Si y es un vértice del tipo $T(1)$, el agente en el es suficiente para limpiar su único hijo en el nivel $i+1$. Después

de que todos los agentes en y son enviados a los hijos de y en el nivel $l+1$, cada vértice de $NT(y)$ es resguardado por un agente y todos los vecinos de y están limpios o resguardados por un agente. Si y es una hoja entonces $NT(y)$ está vacío. Cuando el coordinador llega a él, todos los vecinos de y en el nivel $i+1$ son resguardados y todos los vecinos en el nivel $i-1$ están limpios. ■

Teorema 3.1.4. *El proceso del algoritmo de limpieza desinfecta todos los vértices, además durante su ejecución los vértices descontaminados no pueden recontaminarse.*

Demostración. En el algoritmo de limpieza la descontaminación se lleva a cabo nivel por nivel, así que cuando se llega al nivel d todos los vértices han sido desinfectados. El hecho de que un vértice limpio no vuelva a recontaminarse se debe al lema anterior. ■

Complejidad: Para calcular el número de agentes que se necesitan para limpiar el hipercubo con el algortimo de limpieza primero calculamos el número de agentes que son tomados del conjunto de agentes disponibles antes de realizar la limpieza de un nivel a otro.

Lema 3.1.5. *En el algoritmo de limpieza, una vez que el nivel $l \geq 1$ ha sido limpiado, antes de limpiar el nivel $l+1 \leq d$, $\binom{d}{l+1} - \binom{d}{l} + \binom{d-1}{l-1}$ agentes adicionales son enviados desde la raíz por el coordinador.*

Demostración. En el paso 2.1. del algoritmo de limpieza, $k-1$ agentes adicionales por vértice del tipo $T(k)$ son enviados desde la raíz. Por la Propiedad 3.1.1 sabemos que en el nivel $l \geq 1$ hay $\binom{d-k-1}{l-1}$ vértices del tipo $T(k)$ con $0 \leq k \leq d-l$. Así que en total $\sum_{k=2}^{d-l}(k-1)\binom{d-k-l}{l-1}$ agentes adicionales son enviados desde la raíz al nivel l mientras limpian del nivel l al nivel $l+1$. Notemos que al elegir primero $i = k-1$ y luego $L = l-1$ tenemos que

$$\sum_{k=2}^{d-l}(k-1)\binom{d-k-1}{l-1} = \sum_{i=1}^{d-l-1} i\binom{d-(i+1)-1}{l-1} = \sum_{i=1}^{d-L-2}\binom{i}{1}\binom{d-i-2}{L}.$$

Observemos que dados $a, b \in \mathbb{N}$ tenemos que $\binom{a}{b} = 0$ para $a < b$, entonces

$$\sum_{i=1}^{d-L-2} \binom{i}{1}\binom{d-i-2}{L} = \sum_{i=1}^{d-2} \binom{i}{1}\binom{d-i-2}{L} = \sum_{i=0}^{d-2} \binom{i}{1}\binom{d-i-2}{L}.$$

Recordemos que si $h, m \geq 0$ y $n \geq q \geq 0$, se cumple $\sum_{0\leq k\leq h} \binom{h-k}{m}\binom{q+k}{h} = \binom{h+q+1}{m+n+1}$. Por lo cual se tiene que

$$\sum_{0\leq k\leq h} \binom{h-k}{m}\binom{q+k}{n} = \sum_{i=0}^{d-2} \binom{d-2-i}{m}\binom{q+i}{n},$$

donde $i = k$ y $h = d-2$. Si consideramos $q = 0$, $n = 1$ y $m = L$, entonces

$$\sum_{i=0}^{d-2} \binom{d-2-i}{m}\binom{q+i}{n} = \sum_{i=0}^{d-2} \binom{d-2-i}{L}\binom{i}{1} = \binom{d-2+1}{L+1+1} = \binom{d-1}{L+2}.$$

Por lo que tenemos que

$$\sum_{i=0}^{d-2} \binom{i}{l}\binom{d-i-2}{L} = \binom{d-1}{L+2},$$

por tanto

$$\sum_{i=0}^{d-2} \binom{i}{l}\binom{d-i-2}{L} = \binom{d-1}{l+1} = \binom{d}{l+1} - \binom{d-1}{l} = \binom{d}{l+1} - \binom{d}{l} + \binom{d-1}{l-1}.$$

■

Ahora calculamos el número de agentes usados por el algoritmo de limpieza de un nivel a otro.

Lema 3.1.6. *En el algoritmo de limpieza no son usados más de* $\binom{d}{\frac{d}{2}} + \binom{d-1}{\frac{d}{2}-2} + 1$ *agentes para limpiar del nivel* $l \geq 1$ *al nivel* $l+1 \leq d$.

Demostración. Por inducción sobre el nivel. Veremos que en el nivel 1 se cumple que $d < \binom{d-1}{\frac{d}{2}-2} + \binom{d}{\frac{d}{2}} + 1$ para $d \geq 4$. Primero notemos que

$$\begin{aligned}
\binom{d-1}{\frac{d}{2}-2} + \binom{d}{\frac{d}{2}} + 1 &= \frac{(d-1)!}{\left(\frac{d}{2}-2\right)!\left(\frac{d}{2}+1\right)!} + \frac{d!}{\frac{d}{2}!\frac{d}{2}!} + 1 \\
&= \frac{(d-1)(d-2)\cdots(\frac{d}{2}+1)(\frac{d}{2})\cdots 1}{(\frac{d}{2}-2)!(\frac{d}{2}+1)!} + \frac{d(d-1)(d-2)\cdots\left(\frac{d}{2}\right)\cdots 1}{\left(\frac{d}{2}!\right)^2} + 1 \\
&= \frac{(d-1)(d-2)\cdots(\frac{d}{2}+2)}{(\frac{d}{2}-2)!} + \frac{d(d-1)(d-2)\cdots(\frac{d}{2}+1)}{\frac{d}{2}!} + 1 \\
&= \frac{(d-1)(d-2)\cdots(\frac{d}{2}+2)(\frac{d}{2})(\frac{d}{2}-1) + d(d-1)(d-2)\cdots(\frac{d}{2}+1)}{\frac{d}{2}!} + 1 \\
&= (d-1)(d-2)\cdots\left(\frac{d}{2}+2\right)\left[\frac{\left(\frac{d}{2}\right)\left(\frac{d}{2}-1\right) + d\left(\frac{d}{2}+1\right)}{\frac{d}{2}!}\right] + 1.
\end{aligned}$$

Por otro lado, observemos que

$$\begin{aligned}
d &< d\left(\frac{d}{2}+1\right) \\
&< d\left(\frac{d}{2}+1\right) + \frac{d}{2}\left(\frac{d}{2}-1\right) \\
&< \left[d\left(\frac{d}{2}+1\right) + \frac{d}{2}\left(\frac{d}{2}-1\right)\right]\left[\frac{(d-1)(d-2)\cdots\left(\frac{d}{2}+2\right)}{\frac{d}{2}!}\right] \\
&< \left[d\left(\frac{d}{2}+1\right) + \frac{d}{2}\left(\frac{d}{2}-1\right)\right]\left[\frac{(d-1)(d-2)\cdots\left(\frac{d}{2}+2\right)}{\frac{d}{2}!}\right] + 1.
\end{aligned}$$

Por lo tanto $d < \binom{d-1}{\frac{d}{2}-2} + \binom{d}{\frac{d}{2}} + 1$.

Supongamos que sucede para $l \geq 1$, es decir después de desinfectar l niveles. En el nivel $l \geq 1$ tenemos $\binom{d}{l}$ agentes activos más el coordinador y cada vértice del nivel l está reguardado por un agente, todos los demás agentes son agentes disponibles. Ahora veamos que este algoritmo no utiliza más de $\binom{d}{\frac{d}{2}} + \binom{d-1}{\frac{d}{2}-2} + 1$ agentes para desinfectar el nivel $l+1$. Por el lema anterior sabemos que antes de limpiar el nivel $l+1$ el coordinador colecta $\binom{d}{l+1} - \binom{d}{l} + \binom{d-1}{l-1}$ agentes adicionales y envía exactamente $k-1$ agentes a cada vértice del tipo $T(k)$ del nivel l. Además, por hipótesis de inducción,

cada vértice del nivel l se encuentra en este punto resguardado por un agente desde antes de que el coordinador les asigne un grupo de agentes adicionales a cada uno. Entonces, contando también al coordinador, en total hay $\binom{d}{l}+1+\binom{d}{l+1}-\binom{d}{l}+\binom{d-1}{l-1} = 1+\binom{d}{l+1}+\binom{d-1}{l-1}$ agentes activos en el nivel l. En esta estrategia los $\binom{d-1}{l-1}$ agentes en las hojas no participan en la limpieza del nivel $l+1$, pero los otros $\binom{d}{l+1}$ son apenas suficientes para moverse al nivel $l+1$ en el árbol de transmisión.

Por otro lado, notemos que $\binom{d}{l+1}+\binom{d-1}{l-1} = \binom{d-1}{l+1}+\binom{d-1}{l}+\binom{d-1}{l-1}$, por lo cual, de la Propiedad 1.4.4, se sigue que

$$\begin{aligned}\max_{1\leq l\leq d-1}\left\{\binom{d}{l+1}+\binom{d-1}{l-1}\right\} &= \max_{1\leq l\leq d-1}\left\{\binom{d-1}{l+1}+\binom{d-1}{l}+\binom{d-1}{l-1}\right\}\\ &= \binom{d-1}{(\frac{d}{2}-1)+1}+\binom{d-1}{\frac{d}{2}-1}+\binom{d-1}{(\frac{d}{2}-1)-1}\\ &= \binom{d}{\frac{d}{2}}+\binom{d-1}{\frac{d}{2}-2}.\end{aligned}$$

De esta forma, podemos concluir que no se usan más de $\binom{d}{\frac{d}{2}}+\binom{d-1}{\frac{d}{2}-2}+1$ agentes para desinfectar el nivel $l+1$. ■

Teorema 3.1.7. *El algoritmo de limpieza utiliza $\Theta\left(\frac{n}{\sqrt{\log n}}\right)$ agentes para limpiar el hipercubo, además esta cota es óptima.*

Demostración. La cota inferior está dada por el Teorema 2.2.3. La cota superior se sigue del Lema 3.1.6 al aplicar la aproximación de Stirling [9]. Recordemos que, según la aproximación de Stirling, $\binom{2n}{n} \approx \frac{4^n}{\sqrt{\pi n}}$, por lo tanto

$$\binom{d}{\frac{d}{2}} \approx \frac{4^{\frac{d}{2}}}{\sqrt{\pi\frac{d}{2}}} = \frac{2^d}{\sqrt{\frac{\pi}{2}}\sqrt{d}} = \frac{n}{\sqrt{\frac{\pi}{2}}\sqrt{\log n}}.$$

■

Teorema 3.1.8. *El número total de movimientos ejecutados por los agentes en el algoritmo de limpieza es $O(n\log n)$ y su complejidad es $O(n\log n)$ unidades de tiempo.*

Demostración. Para contar el número total de movimientos ejecutados tomamos en cuenta los movimientos realizados por los agentes y también los movimientos del coordinador.

Movimientos ejecutados por los agentes: Un agente necesita $2l$ movimientos para llegar de la raíz a una hoja del nivel l y regresar a la raíz. Por la propiedad 1 sabemos que hay $\binom{d-1}{l-1}$ hojas en el nivel l. Así que en total hay $\sum_{l=1}^{d} 2l\binom{d-1}{l-1}$ movimientos ejecutados por los agentes. Para calcular esta cantidad primero recordemos que $\binom{d}{l}$ es el número de vértices en el nivel l. Si sumamos sobre todos los niveles del hipercubo obtenemos: $\sum_{l=0}^{d}\binom{d}{l} = 2^d = n$, ahora observemos que

$$\sum_{l=1}^{d}\binom{d-1}{l-1} = \sum_{l=0}^{d-1}\binom{d-1}{l} = \binom{d-1}{0}+\binom{d-1}{1}+\cdots+\binom{d-1}{d-1} = 2^{d-1} = \frac{2^d}{2} = \frac{n}{2}.$$

Ahora, desarrollemos la siguiente suma

$$\begin{aligned}\sum_{l=1}^{d} l\binom{d-1}{l-1} &= 1\binom{d-1}{0} + 2\binom{d-1}{1} + 3\binom{d-1}{2} + \cdots + \left(\frac{d}{2}-1\right)\binom{d-1}{\frac{d}{2}-2} \\ &\quad + \frac{d}{2}\binom{d-1}{\frac{d}{2}-1} + \left(\frac{d}{2}+1\right)\binom{d-1}{\frac{d}{2}} + \left(\frac{d}{2}+2\right)\binom{d-1}{\frac{d}{2}+1} + \cdots \\ &\quad + (d-2)\binom{d-1}{d-3} + (d-1)\binom{d-1}{d-2} + d\binom{d-1}{d-1}.\end{aligned}$$

Recordemos que $\binom{r}{s} = \binom{r}{r-s}$ para $0 \leq s \leq r$. Por lo que, agrupando los términos de la expresión anterior por parejas, obtenemos:

$$\begin{aligned}\sum_{l=1}^{d} l\binom{d-1}{l-1} &= (d+1)\binom{d-1}{0} + (d+1)\binom{d-1}{1} + \cdots + (d+1)\binom{d-1}{\frac{d}{2}-1} \\ &= (d+1)\sum_{l=0}^{\frac{d}{2}-1}\binom{d-1}{l}.\end{aligned}$$

Recordemos que $\sum_{l=0}^{d-1}\binom{d-1}{l} = 2^{d-1} = \frac{n}{2}$, y además observemos que $\sum_{l=0}^{d-1}\binom{d-1}{l} = 2\sum_{l=0}^{\frac{d}{2}-1}\binom{d-1}{l}$ para d par, por lo cual tenemos $2\sum_{l=0}^{\frac{d}{2}-1}\binom{d-1}{l} = 2^{d-1}$. De lo anterior se

sigue que $\sum_{l=0}^{\frac{d}{2}-1} \binom{d-1}{l} = 2^{d-1}2^{-1} = 2^{d-2} = \frac{2^d}{2^2} = \frac{n}{4}$, lo cual implica que $\sum_{l=1}^{d} l\binom{d-1}{l-1} = (d+1)2^{d-2} = \frac{n}{4}(\log n + 1)$. Finalmente, obtenemos que $\sum_{l=1}^{d} 2l\binom{d-1}{l-1} = \frac{n}{2}(\log n + 1)$ y así podemos concluir el orden $O(n \log n)$ para los movimientos ejecutados por los agentes.

Movimientos ejecutados por el coordinador:

1. Primero observemos que del nivel 1 llegar a la raíz nos toma un paso, del nivel 2 a la raíz nos toma dos pasos, y así en general para el nivel l con $1 \leq l \leq d$ toma l pasos llegar a la raíz. Además por construcción del hipercubo notamos que al estar en la iteración $d-1$ del algoritmo de limpieza estamos en el nivel $d-2$ y nos lleva $d-2$ pasos regresar a la raíz, ya que cuando el agente está en el nivel l donde $1 \leq l \leq d-2$ hay al menos un vértice del tipo $T(2)$ y en el nivel $d-1$ ya no es necesario regresar a la raíz por más agentes pues los vértices que quedan son del tipo $T(0)$ y $T(1)$, por tanto en general decimos que ir a la raíz por más agentes nos toma $\sum_{l=1}^{d-2} l = \frac{(d-2)(d-1)}{2} = O(\log^2 n)$ pasos, ya que $\frac{(d-2)(d-1)}{2} = \frac{d^2-3d+2}{2} = O(d^2) = O((\log n)^2)$ donde $d = \log n$.

2. Para contar el número de pasos que nos toma llegar al primer vértice de cada nivel es muy sencillo notar que para llegar al nivel 1 nos toma un paso, al nivel 2 nos toma dos pasos y así en general para el nivel l con $1 \leq l \leq d$, toma l pasos llegar a dicho vértice aparte existe un único primer vértice en cada nivel, por construcción del hipercubo en orden lexicográfico. Por tanto este número de pasos es

$$\sum_{l=1}^{d} l = \frac{d(d+1)}{2} = O(\log^2 n).$$

3. Ahora contaremos el número de pasos que usamos al desplazarnos a través de los niveles para llegar de un vértice a otro. Para esto, recordemos que estamos en un hipercubo y, por lo tanto, para llegar de un vértice a otro necesitamos a lo más $2l$ pasos.

Sabemos que hay $\binom{d}{l}$ vértices en el nivel l por tanto el número de pasos es menor o igual que

$$\sum_{l=1}^{d-1} 2l\binom{d}{l} = 2\sum_{l=1}^{d-1} l\binom{d}{l} = 2\left[\sum_{l=1}^{\frac{d}{2}-1} l\binom{d}{l} + \sum_{l=\frac{d}{2}}^{d-1} l\binom{d}{l}\right] \leq 2\left[2\sum_{l=1}^{\frac{d}{2}} l\binom{d}{l}\right]$$
$$= 4\sum_{l=1}^{\frac{d}{2}} l\binom{d}{l} = O(n\log n).$$

4. Para ir con cada agente a limpiar un vértice en el siguiente nivel en el árbol de transmisión y después regresar a la raíz: Cada arista en el árbol generador de transmisión es recorrida dos veces por el coordinador, entonces tenemos que hay $2(2^d - 1) = 2(n - 1)$ movimientos (dos movimientos por cada arista del árbol).

Recordemos que el ambiente en el que trabajamos es asincrónico, así que consideraremos la complejidad ideal del tiempo para la estrategia de limpieza (es decir, asumiremos que le toma una unidad de tiempo a un agente recorrer una arista). Si observamos que el proceso de limpieza se lleva a cabo secuencialmente por el coordinador, entonces notamos que el tiempo requerido es igual al número de movimientos ejecutados por el coordinador.

Nota: Observaremos brevemente el caso en el que la dimensión del hipercubo es impar. Ya que el algoritmo no depende en si d es par o impar, la corrección se mantiene. Lo único que se ve afectado es el número máximo de agentes empleados. De hecho en la prueba del Lema 3.1.6 hemos demostrado que el número de agentes empleados es $\max_{1\leq l\leq d-1}\left\{\binom{d}{l+1} + \binom{d-1}{l-1}\right\} + 1$. Cuando d es par, ya vimos que este número corresponde al número de vértices $\binom{d}{\frac{d}{2}}$ en el nivel máximo más el número de hojas $\binom{d-1}{\frac{d}{2}-2}$, en el nivel anterior, más 1. Cuando d es impar hay dos niveles centrales ($\frac{d+1}{2}$ y $\frac{d-1}{2}$) con el mismo número de vértices en el árbol generador. En este caso $\max_{1\leq l\leq d-1}\{\binom{d}{l+1} + \binom{d-1}{l-1}\} + 1$ corresponde al número de vértices en el nivel $\frac{d+1}{2}$, $\binom{d}{\frac{d+1}{2}}$, más el número de hojas $\binom{d-1}{\frac{d+1}{2}-1} = \binom{d-1}{\frac{d-1}{2}}$ en el nivel anterior, más 1. Este número claramente sigue siendo $O(\frac{n}{\sqrt{\log n}})$. ■

Capítulo 4

Estrategia de Limpieza sin Coordinador

4.1. Modelo de Visibilidad: Estrategia de Tiempo Óptimo

Ahora veamos una solución al problema de descontaminación en un modelo donde los agentes pueden "ver" el estado de todos sus vecinos, ya sea que esten limpios, resguardados o contaminados. De hecho haremos la siguiente suposición: Un agente localizado en el vértice x puede ver el estado de sus vecinos $N(x)$.

Veremos que el hecho de tener visibilidad permite que los agentes decidan su próximo movimiento únicamente con las bases de sus conocimientos locales y sin la necesidad de ser guiados por un coordinador. Esta cualidad nos permite reducir el tiempo de complejidad, sin embargo ocasiona un incremento en el número de agentes.

4.2. Propiedades Básicas

Introduciremos algunas propiedades básicas que necesitaremos para este algoritmo. Denotaremos por C_i al conjunto de vértices cuya coordenada más significativa se encuentra en la posición i. Por tanto C_d es exactamente el conjunto de todas las hojas del árbol de transmisión.

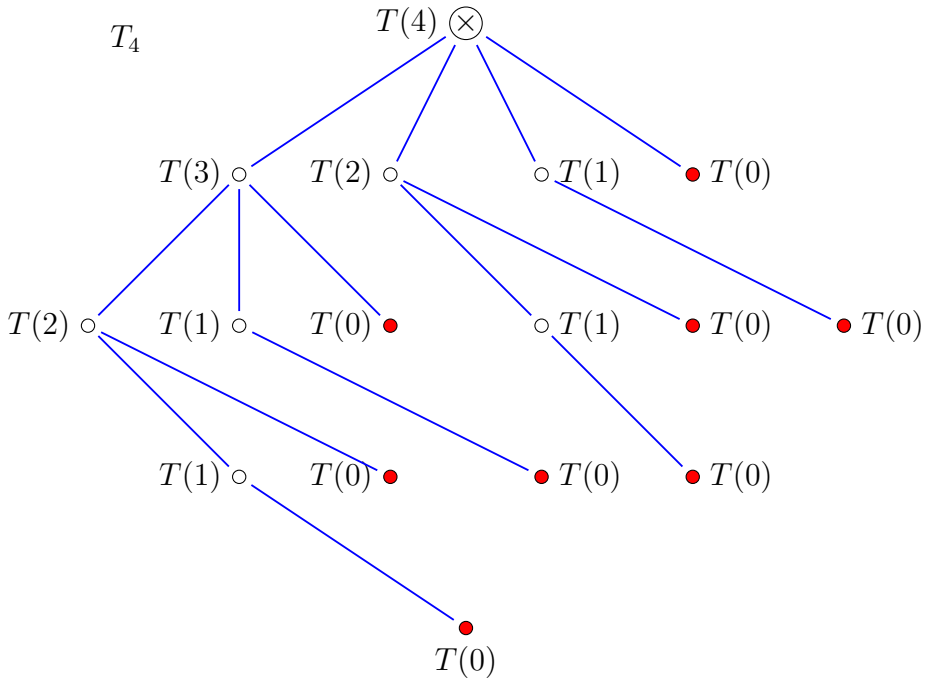


Figura 4.1: En morado podemos observar el conjunto de vértices que son hojas en el árbol generador del hipercubo 4 dimensional.

Propiedad 4.2.1. . *El conjunto C_0 contiene exactamente un vértice; el conjunto C_i, para $0 < i \leq d$, contiene 2^{i-1} vértices.*

Demostración. Los vértices en C_i tienen su coordenada más significativa en la posición i de la n-ada, es decir, la coordenada i-ésima es igual a 1 y para cada $j > i$, todas las coordenadas son iguales a 0. Para cualquier coordenada menor que i tenemos dos posibilidades: 0 ó 1; como hay $i - 1$ lugares para los cuales se puede hacer esta elección, hay 2^{i-1} vértices en C_i. ■

Propiedad 4.2.2. *Sea x cualquier vértice en C_i con $i > 0$. Un vecino pequeño de x está en C_j, donde $j < i$, y los demás vecinos pequeños, si es que existe alguno, están en C_i. Los vecinos grandes de x, si es que alguno existe, están en C_k, donde $k > i$.*

Demostración. Por definición tenemos que la coordenada más significativa del vértice x se encuentra en la i-ésima entrada. Las etiquetas de sus vecinos difieren en sólo una entrada.

Si un vecino y es diferente de x en la entrada l, con $l < i$, entonces y es un vecino pequeño de x y entonces y tiene un 1 en la i-ésima coordenada, por lo tanto y se encuentra en C_i también. Si y difiere de x en la i-ésima entrada, entonces la coordenada más significativa de y debe estar en la j-ésima posición, donde $j < i$; entonces y se encuentra en C_j, y por definición y también es un vecino pequeño de x. Dado que por construcción únicamente existe un vecino que difiere de x en la i-ésima entrada, tenemos que un vecino pequeño de x está en C_j, donde $0 \leq j < i$, y que los demás vecinos pequeños están en C_i. Si un vecino y es distinto a x en la entrada k-ésima de su coordenada, con $k > i$, entonces la k-ésima entrada debe ser 1, ya que la coordenada más significativa de x está en la i-ésima posición. Por definición y es un vecino grande de x, y se encuentra en C_k donde $k > i$. ■

Observemos que el vértice $(0, 0, \ldots, 0)$ está en C_0 y no tiene vecinos pequeños, además el vértice $(0, 0, \ldots, 0, 1)$ está en C_1 y tiene un vecino pequeño en C_0, pero no vecinos pequeños en C_1, es por esto que en la siguiente propiedad sólo consideraremos a los vértices x en C_i con $i > 2$.

Propiedad 4.2.3. *Sea x cualquier vértice en C_i donde $i > 2$. Debe existir al menos un vecino pequeño y de x tal que y esté en C_i y un vecino pequeño z de y tal que z esté en C_{i-1}.*

Demostración. Sea x cualquier vértice en C_i, $i > 2$. Como $x \in C_i$, sabemos que la coordenada más significativa de x está en la posición i.

- Caso 1. Si la entrada $i-1$ de x es igual a 0, consideremos a y, un vecino pequeño de x que difiera de x en exactamente la coordenada $i - 1$, entonces y tiene 1 en la coordenada $i - 1$ y como sólo difiere de x en esta coordenada, y tiene 1 en la entrada i (por lo que la coordenada más significativa de y es la i-ésima), por lo tanto $y \in C_i$.

- Caso 2. Si la entrada $i-1$ de x es 1, consideraremos a y, un vecino pequeño de x que difiera de x en una coordenada estrictamente menor a $i-1$. Como y sólo difiere de x en una entrada menor a $i-1$ entonces y tiene 1 en las entradas i e $i-1$ (pues x tiene 1 en esas posiciones), entonces $y \in C_i$.

Sea z un vecino pequeño de y que difiere de y en la i-ésima coordenada entonces z tiene 0 en la i-ésima coordenada y además y tiene 1 en la $(i-1)$-coordenada, por tanto la entrada más significativa de z está en la $(i-1)$-ésima coordenada y por lo tanto $z \in C_{i-1}$ ■

Estrategia de Limpieza

Todos los agentes se encuentran inicialmente en la raíz, es decir en el vértice fuente del árbol de transmisión, todos ellos son idénticos y autónomos y siguen las mismas normas locales. Los agentes se mueven a lo largo del árbol de transmisión como en el modelo anterior, pero pueden llevar a cabo esta estrategia sin tener un líder o coordinador. De hecho pueden proceder inmediatamente a desinfectar a sus hijos (vecinos grandes) en el árbol de transmisión cuando "ven" que sus vecinos pequeños están limpios o resguardados.

4.2.1. Algoritmo de Limpieza con Visibilidad

Normas para los agentes en el vértice x del tipo $T(k)$, donde $0 \leq k \leq d$.

1. Esperar hasta que 2^{k-1} agentes estén en x.
2. Si $k < d$ entonces esperar a que todos los vecinos pequeños de x estén limpios o resguardados.
3. Un agente se mueve hacia un vecino grande del tipo $T(0)$; 2^{i-1} agentes se mueven a cada uno de los vecinos grandes del tipo $T(i)$, para $0 < i < k$; si ya no hay vecinos grandes terminan.

En esta estrategia la limpieza de los vértices no se hace secuencialmente, de hecho se puede llevar a cabo independientemente.

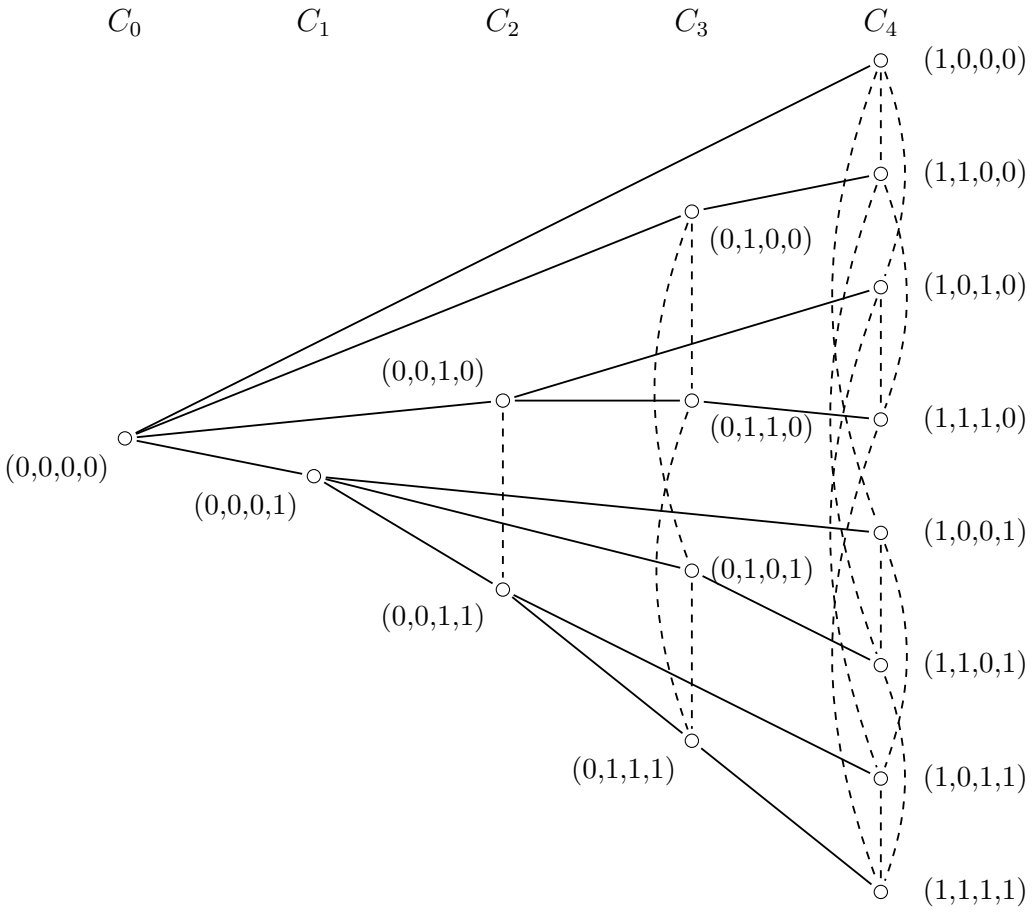


Figura 4.2: Clasificación de los vértices respecto a C_i, con $0 \leq i \leq 4$, en H_4.

Corrección y Complejidad

Teorema 4.2.4. *El número total de agentes que se necesitan para desinfectar el hipercubo H_d usando el algoritmo de Limpieza con Visibilidad es $\frac{n}{2}$.*

Demostración. Por definición el algoritmo de limpieza con visibilidad envía un agente del nivel 0 al nivel 1 para $T(0)$ y envía 2^{i-1} agentes para cada $T(i)$, para un total de $1 + \sum_{i=1}^{d-1} 2^{i-1} = 1 + \sum_{i=0}^{d-2} 2^i = 2^{d-1} = \frac{n}{2}$ agentes.

Además un vértice del tipo $T(k)$ recibe 2^{k-1} agentes y 2^{k-1} es exactamente el número de agentes que se necesitan para continuar con la estrategia de limpieza.

De hecho $2^{k-1} = 1 + \sum_{i=1}^{k-1} 2^{i-1}$, por tanto con $\frac{n}{2}$ agentes, la estrategia se puede completar. ■

Ahora demostraremos que el algoritmo de Limpieza con Visibilidad es correcto, en otras palabras, la red se limpia y una vez hecha la desinfección ésta no se puede recontaminar.

Lema 4.2.5. *En el Algoritmo de Limpieza con Visibilidad cuando los agentes dejan un vértice en C_i (es decir, lo dejan sin resguardo), todos sus vecinos pequeños están desinfectados o resguardados.*

Demostración. Consideremos un vértice x en C_i. Por la estrategia de limpieza, cuando un agente llega al vértice x, éste desinfecta el vértice. Por lo ya demostrado en el teorema anterior, todos los vértices tendrán suficientes agentes para continuar con el proceso de limpieza, y por la segunda norma del algoritmo, los agentes en x se mueven hacia los vecinos grandes únicamente cuando todos los vecinos pequeños están limpios o resguardados. ■

Teorema 4.2.6. *Durante el proceso de desinfección con el Algoritmo de Limpieza con Visibilidad los agentes limpian todos los vértices, y un vértice limpio no puede recontaminarse.*

Demostración. Por la estrategia de limpieza las aristas y los vértices recorridos por los agentes forman el árbol de transmisión. Todos los vértices son visitados por un agente (al menos una vez). El hecho de que un vértice desinfectado no se recontamine se sigue directamente del lema anterior. ■

Ahora consideraremos la complejidad del algoritmo.

Teorema 4.2.7. *El desinfectar la red con el Algoritmo de Limpieza con Visibilidad toma $\Theta(\log n)$ unidades de tiempo.*

Demostración. Observemos que, por el Teorema 4.2.6, basta demostrar que en el tiempo i todos los vértices en C_i están resguardados. En el tiempo $i = 0$, todos los agentes se encuentran en el vértice $(0, 0, \ldots, 0)$, por lo cual está resguardado. Ahora, demostraremos por inducción para $i \geq 1$, que al tiempo i todos los vértices en C_i están resguardados y que cualquier vértice resguardado en C_j para algún $i < j \leq d$ queda resguardado al menos hasta el siguiente tiempo.

Primero notemos que no hay ningún agente en otro vértice en el tiempo 0, así que sólo los agentes en C_0 se pueden mover en este tiempo. Por la estrategia de limpieza con visibilidad, los agentes limpian la fuente y después se desplazan a limpiar a sus d vecinos grandes, los cuales están en C_j para $0 < j \leq d$. Por lo anterior, todos los vecinos de $(0, 0, \ldots, 0)$, son los únicos vértices resguardados al tiempo $i = 1$, en particular, el vértice $(0, \ldots, 0, 1)$ está resguardado, el cual es el único elemento de C_1. Sea x_j el vecino de $(0, 0, \ldots, 0)$ cuya j-ésima entrada es igual a 1 y las demás entradas igual a 0, para $j \geq 2$. Consideremos al vértice y_j, cuyas j-ésima y $(j-1)$-ésima entradas sean igual a 1 y todas las demás entradas igual a 0. Dicho vértice y_j es un vecino pequeño de x_j que no está limpio, ni resguardado al tiempo $i = 1$. Por lo cual, x_j se mantendrá resguardado (al menos) hasta el tiempo 2. Por tanto, la afirmación se cumple para $i = 1$.

Supongamos que nuestra afirmación se cumple para cualquier tiempo menor o igual a i, donde $i \geq 1$. Demostraremos que se cumple para el tiempo $i + 1$. Sea x un vértice arbitrario de C_{i+1}. Por la Propiedad 4.2.2, exactamente un vecino pequeño y de x está en C_k con $k \leq i$. De la hipótesis inductiva se tiene que dicho vecino y ha sido resguardado en el tiempo k, y además, por la hipótesis inductiva, el Lema 4.2.5 y el Teorema 4.2.6, todos los vecinos pequeños de y se encuentran limpios o resguardados al tiempo k. Por lo cual, al tiempo $k+1$, el vértice x está resguardado, y de nuevo por la hipótesis inductiva, x se mantendrá resguardado del tiempo $k+1$ hasta el tiempo $i + 1$.

Ahora veremos que por el algoritmo de limpieza, los demás agentes en C_j para $i+1 < j \leq d$ no se pueden mover ya que uno o más de sus vecinos pequeños no están resguardados. Consideremos cualquier vértice u en C_j en el cual hay agentes, por la Propiedad 4.2.3 existe un vecino pequeño v de u que está en C_j y también un vecino

pequeño w de v que está en C_{j-1}, donde $j-1 \geq i+1$. Sabemos que los agentes en v vienen de su vecino pequeño w que está en C_{j-1}. Si $j-1 = i+1$, o en otras palabras, si w está en C_{i+1}, los agentes en w se mueven al tiempo $i+1$. Así que en el momento $i+1$ no hay agentes en v aún. Si $j > i+1$, aunque haya agentes en w, por la hipótesis de inducción, no se mueven antes del tiempo $i+1$. Por lo tanto en cualquier caso, v no está resguardado al tiempo $i+1$ y los agentes en u no se pueden mover, ya que al menos uno de sus vecinos pequeños no está resguardado.

Finalmente, la complejidad es claramente óptima ya que $\log n$ es el diámetro del hipercubo y los agentes están inicialmente situados en el mismo vértice. ■

Ahora calcularemos el número total de movimientos efectuados por los agentes.

Teorema 4.2.8. *El número de movimientos efectuados por los agentes en el Algoritmo de Limpieza con Visibilidad para la desinfección es $O(n \log n)$.*

Demostración. Todos los agentes comienzan en el vértice $(0, 0, \ldots, 0)$ y cada uno termina en una hoja. En total hay $\binom{d-1}{l-1}$ hojas en el nivel $l > 0$, así que el número total de movimientos es $\sum_{l=1}^{d} l\binom{d-1}{l-1} = O(n \log n)$. ■

Capítulo 5

Estrategias de Limpieza sin Coordinador y con Clonación

5.1. Visibilidad y Clonación; Estrategia de Tiempo y Movimiento Óptimos

La clonación es la capacidad de un agente de crear copias de si mismo. Ahora consideraremos el modelo de visibilidad donde los agentes tienen la capacidad de clonarse. Mostraremos cómo para este modelo no necesitamos un coordinador para controlar la desinfección, de hecho todos los agentes pueden seguir el mismo algoritmo de forma autónoma. La clonación permite que los agentes ejecuten un menor número de movimientos.

Inicialmente un agente se situa en el vértice fuente, desinfecta la fuente y clona $d - 1$ nuevos agentes. De esta manera d agentes se encuentran disponibles en la fuente. Después un agente por arista incidente es enviado para limpiar cada uno de los d vecinos. Cuando el agente llega al vértice, éste lo limpia y después ejecuta el siguiente algoritmo.

5.1.1. Algoritmo de Limpieza con Visibilidad y Clonación

Normas para los agentes en el vértice x:

1. Esperar a que todos los vecinos pequeños de x se encuentren limpios o resguardados.
2. Clonar suficientes agentes para limpiar a los vecinos grandes de x y después enviar un agente por arista incidente a éste para desinfectar cada uno de sus vecinos grandes. Si todos los vecinos pequeños están resguardados o limpios y ya no hay vecinos grandes (es decir, se encuentra en una hoja) termina.

Teorema 5.1.1. *Durante el proceso de limpieza con visibilidad y clonación los agentes desinfectan todos los vértices y un vértice limpio no puede recontaminarse.* ■

Se omite la demostración del teorema anterior debido a que es análoga a la del Teorema 4.2.6.

Teorema 5.1.2. *La desinfección de la red con el Algoritmo de Limpieza con Visibilidad y Clonación requiere* $\log n$ *unidades de tiempo,* $\frac{n}{2}$ *agentes y* $n-1$ *movimientos. La complejidad y el número de movimientos son óptimos.*

Demostración. La prueba de las unidades de tiempo es análoga a la del Teorema 4.2.7. Veamos primero el número de agentes que se utilizan para ejecutar este algoritmo. Por la estrategia de limpieza, el número de agentes que se utilizan es el número de agentes que terminan el algoritmo, entonces contaremos este número. Por la forma en que está definida la estrategia de limpieza, los vértices y las aristas recorridas por los agentes forman el árbol de transmisión; cada vértice es visitado exactamente por un agente, el cual se clona si hay vecinos grandes que limpiar. Por el Algoritmo de Limpieza con Visibilidad y Clonación, un agente en un vértice termina el algoritmo sólo si el vértice no tiene vecinos grandes (es una hoja en el árbol de transmisión). Recordemos que el número de hojas en T_d es 2^{d-1}. Cada uno de estos 2^{d-1} vértices es visitado exactamente por un agente, por lo tanto el número total de agentes usados en el algoritmo es

$$2^{d-1} = \frac{2^d}{2} = \frac{n}{2}.$$

Finalmente, el cálculo del número de movimientos se sigue del hecho de que las aristas y los vértices recorridos por los agentes forman el árbol de transmisión y que cada arista en el árbol de transmisión es recorrida únicamente por un agente, recordando que el número de aristas en el árbol de transmisión es $2^d - 1$ tenemos que los agentes ejecutan $n - 1$ movimientos ($2^d = n$).

La complejidad del tiempo es óptima porque $\log n$ es el diámetro del hipercubo, y todos los agentes comienzan el algoritmo en el mismo vértice, el vértice fuente. El número de movimientos también es óptimo, pues todos los vértices deben ser visitados. ■

5.2. Localidad, Clonación y Sincronicidad: Estrategia de Tiempo y Movimiento Óptimos

Ahora veremos que el resultado obtenido previamente para el modelo de visibilidad con clonación se puede obtener también en el modelo local cuando la clonación está disponible y el sistema es sincrónico. De hecho, vamos a describir una estrategia para el modelo local que explota la sincronicidad, para así obtener una complejidad óptima en tiempo, y la clonación para obtener un número óptimo de movimientos, esto lo lograremos a cuestas del número de agentes usados.

Nuestro acercamiento lo basamos en la observación de que, aunque nuestros agentes no tengan visibilidad, se pueden seguir moviendo autónomamente gracias a la sincronicidad del sistema. Explotaremos la sincronicidad usando una estrategia muy parecida a la del Algoritmo de Limpieza con Visibilidad, pero sin la necesidad de asumir la visibilidad. En lugar de esperar a que todos los vecinos pequeños estén limpios o resguardados, los agentes en un vértice esperan una "cantidad" o un "lapso" apropiado de tiempo antes de moverse para limpiar a sus vecinos grandes. Recordemos que $m(x)$ denota la coordenada más significativa de x. En el modelo sincrónico, los agentes en x se pueden mover a los vecinos grandes cuando el tiempo sea $t = m(x)$, ya que ellos implícitamente saben que en ese tiempo todos los vecinos pequeños de x están limpios o resguardados. En el tiempo 0, un agente es situado en el vértice fuente, éste limpia la fuente y clona $d - 1$ nuevos agentes. Un agente por cada arista incidente en el vértice es enviado a desinfectar cada uno de los d vecinos. En el tiempo i, un agente llega a un vértice, lo limpia y después ejecuta el siguiente algoritmo.

5.2.1. Algoritmo de Limpieza con Clonación y Sincronicidad

Normas para los agentes en el vértice x:

1. Esperar hasta que $t = m(x)$.
2. Clonar suficientes agentes para limpiar a los vecinos grandes de x y después enviar un agente por cada arista incidente a limpiar cada uno de sus vecinos grandes. Si ya no tienen vecinos grandes terminan.

Corrección y Complejidad

La corrección se sigue del siguiente resultado.

Lema 5.2.1. *En el tiempo i, donde $0 \leq i \leq d$, del Algoritmo de Limpieza con Clonación y Sincronicidad, hay un agente en cada vértice de C_i. Cada vértice clona otros agentes y limpia los vecinos grandes. Todos los vértices en C_j con $0 \leq j \leq i$, están limpios o resguardados. En el tiempo d, todos los agentes en los vértices de C_d han terminado.*

Demostración. Procederemos por inducción sobre el tiempo i. Notemos que en el tiempo $i = 0$ únicamente el vértice fuente está en C_0 y la estrategia de limpieza pone exactamente un agente en él. Este agente limpia el vértice, se clona y limpia a sus vecinos, así que no puede ocurrir una recontaminación y en el tiempo 1 todos los vértices en C_0 y C_1 están limpios o resguardados.

Ahora supongamos que en el tiempo i, para $0 \leq i \leq d-1$, hay un agente en cada vértice del tipo C_i, y que todos los vértices en C_j donde $0 \leq j \leq i$, están limpios o resguardados.

Ahora demostraremos que en el tiempo $i+1$, hay un agente en cada vértice de C_{i+1}. Dicho agente se clona y mueve a un vecino grande (si es que lo tiene) o termina. Además todos los vértices en C_{j+1}, para $0 \leq j \leq i$ están limpios o resguardados. Sea x cualquier vértice en C_{i+1}, por la Propiedad 4.2.2, exactamente un vecino pequeño de x, al cual llamaremos z está en C_j, para algún $j \leq i$, y todos los demás vecinos pequeños de x, si es que tiene, están en C_{i+1}. Por la hipótesis de inducción, en el

tiempo j, un agente es enviado desde z hasta x y debe esperar en x hasta el tiempo $t = i+1$. En este tiempo, si $i+1 \leq d-1$ entonces el agente clona suficientes agentes y desinfecta a sus vecinos; en otro caso, $i + 1 = d$ y el agente está en C_d, lo que quiere decir que se encuentra en una hoja, así que termina el algoritmo. Además, en el tiempo $i + 1$, cada vértice en C_{i+1} está resguardado por un agente. Los únicos vecinos contaminados de x son sus vecinos grandes. El agente en x clona suficientes nuevos agentes, limpia a sus vecinos y el vértice x cambia de un estado de resguardo a un estado limpio, junto con sus vecinos pequeños en C_{i+1}. Por tanto los vértices en C_{j+1}, para $0 \leq j \leq i$, no se pueden recontaminar. ■

El siguiente teorema establece que la estrategia de limpieza es correcta, esto es, la red está limpia y una vez que un vértice ha sido desinfectado, éste no podra recontaminarse. Es claro que la demostración correspondiente se sigue del lema anterior.

Teorema 5.2.2. *Durante el proceso de desinfección del Algoritmo de Limpieza con Clonación y Sincronicidad los agentes limpian todos los vértices y un vértice limpio no podrá recontaminarse.* ■

Respecto a la complejidad tenemos el siguiente teorema.

Teorema 5.2.3. *La desinfección de la red con el Algoritmo de Limpieza con Clonación y Sincronicidad requiere* $\log n$ *unidades de tiempo,* $\frac{n}{2}$ *agentes y* $n-1$ *movimientos. La complejidad del tiempo y los movimientos es óptima.*

Demostración. La cota en las unidades de tiempo se sigue del Lema 5.2.1 y la prueba de la cota del número de agentes y movimientos es análoga a la del Teorema 5.1.2. ■

Capítulo 6

Conclusión

Con este trabajo analizamos tres estrategias de descontaminación de hipercubos n-dimensionales, donde nuestro hipercubo es de grado n par, con $n \geq 4$.

La primera estrategia que vimos es la Estrategia de Limpieza con Coordinador, en ésta observamos que los agentes tienen localidad, es decir que los agentes únicamente cuentan con información local, por lo que necesitan ayuda de un lider, alguién que los guíe a traves del árbol generador de transmisión creado para viajar por el hipercubo para descontaminarlo. Por lo que al aplicar esta estrategia llegamos al número óptimo de agentes $\Theta\left(\frac{n}{\sqrt{\log n}}\right)$, además vimos que el número total de movimientos ejecutados por los agentes en el algoritmo de limpieza es $O(n \log n)$ y su complejidad es $O(n \log n)$ unidades de tiempo.

Después vimos la Estrategia de Limpieza sin Coordinador, en ésta los agentes cuentan con la capacidad de visualizar el estado de sus vértices vecinos, por lo que ya no necesitan un lider que los guíe, entonces en esta estrategia contamos con agentes autónomos, pero también necesitamos más agentes en total $\frac{n}{2}$, de igual forma que en el algoritmo anterior requerimos $O(n \log n)$ movimientos, pero alcanzamos la complejidad óptima de tiempo con $\log n$ pasos.

Finalmente para la estrategia de Limpieza sin Coordinador y con Clonación utilizamos dos algoritmos distintos el Algoritmo de Limpieza con Visibilidad y Clonación y el Algoritmo de Limpieza con Clonación y Sincronicidad. Con ambos algoritmos no requerimos de un coordinador por lo que los agentes son autónomos, en el caso del Algoritmo de Limpieza con Visibilidad y Clonación gracias a la visibilidad, y en el caso del otro algoritmo gracias a la sincronicidad pues con ésta cada paso que se da en el algoritmo es instantáneo y toma una sola unidad de tiempo para cada agente

moverse de un vértice a otro vértice vecino, por lo que obtenemos $\log n$ unidades de tiempo en $n-1$ movimientos en cada algoritmo por lo que ambos algoritmos obtienen complejidad óptima tanto en tiempo como en movimientos, el costo de este beneficio se ve en el número de agentes requeridos el cual aumenta quedando en $\frac{n}{2}$.

En la siguiente tabla se muestran las distintas funciones de complejidad para cada uno de los algoritmos que estudiamos. Podemos observar que en ningún caso se alcanza la complejidad óptima para todos los parámetros simultáneamente, por lo que se deben tomar en consideración los recursos disponibles en la situación a resolver para elegir la más adecuada.

	Agentes	Tiempo	Movimientos
Coordinador	$O(\frac{n}{\sqrt{\log n}})$	$O(n \log n)$	$O(n \log n)$
Visibilidad	$n/2$	$\log n$	$O(n \log n)$
Clonación y visibilidad	$n/2$	$\log n$	$n-1$
Clonación y sincronicidad	$n/2$	$\log n$	$n-1$

Bibliografía

[1] P. Flocchini, M.J. Huang, F.L. Luccio. Capturing an Intruder in the Hypercube by Mobile Agents. *Discrete Mathematics* **309(8)** (2009) 1971-1985.

[2] L. Barriere, P. Flocchini, P. Fraigniaud, N. Santoro. Capture of an intruder by mobile agents. *Proceedings of the 14th ACM Symposium on Parallel Algorithms and Architectures (SPAA)*. (2002) 200-209.

[3] J.A. Bondy, U.S.R. Murty. *Graph Theory*. Springer, 2008.

[4] L.S. Chandran, T. Kavitha. The treewidth and pathwidth of hypercubes. *Discrete Mathematics* **306** (2006), 359–365.

[5] Y. Daadaa, A. Jamshed, M. Shabbir. Network Decontamination with a Single Agent. *Graphs and Combinatory* **32**(2) (2016), 559–581.

[6] R. Diestel. *Graph Theory*. Springer, 2005.

[7] P. Flocchini, F. L. Luccio, L. X. Song. Size optimal strategies for capturing an intruder in mesh networks. *Proceedings of the 2005 International Conference on Communications in Computing, 200-206*. CSREA Press, 2005.

[8] P. Flocchini, B. Mans, N. Santoro. Tree Decontamination with Temporary Immunity. *Proceedings of the 19th. International Symposium on Algorithms and Computation (ISAAC)* **5369** 330–341. Springer, 2008.

[9] O. Hijab. *Introduction to Calculus and Classical Analysis*. Springer, 2007.

[10] D. Jungnickel. *Graphs, Networks and Algorithms*. Springer, 2013.

[11] L.M. Kirousis, C.H. Papadimitriou. Searching and Pebbling. *Theoretical Computer Science* **47** (1986), 205–218.

[12] F. Luccio, L. Pagli, N. Santoro. Network Decontamination in Presence of Local Immunity. *International Journal of Foundations of Computer Science* **18**(3) (2007), 457–474.

[13] T. D. Parsons. Pursuit-evasion in a graph. *Theory and Applications of Graphs* (1976), 426–441.

Índice alfabético