INSTITUTO POLITÉCNICO NACIONAL

CENTRO DE INVESTIGACIÓN EN COMPUTACIÓN

ALGORITMOS DE APROXIMACIÓN DISTRIBUIDOS PARA LA DISEMINACIÓN DE INFORMACIÓN EN MANETS 3D

TESIS PRESENTADA POR ARIANA CRUZ TREJO

PARA OBTENER EL GRADO DE MAESTRA EN CIENCIAS DE LA COMPUTACIÓN

DIRECTORES DE LA TESIS:

DR. ROLANDO MENCHACA MÉNDEZ

DR. FRANCISCO JAVIER ZARAGOZA MARTÍNEZ

2012

Laboratorio de Comunicaciones y Redes de Computadoras

Agradecimientos

Este trabajo no hubiera sido posible sin el apoyo, las enseñanzas y la supervisión de mis directores de tesis, el Dr. Rolando Menchaca Méndez y el Dr Francisco Javier Zaragoza Martínez a quienes les agradezco muy profundamente por su disposición para que yo concluyera esta etapa de la mejor manera posible.

A los profesores que formaron parte de mi comité tutorial: el Dr. Marco Antonio Moreno Armendáriz, Dr. Rene Luna García, M. en C. Germán Téllez Castillo y el Dr. Marco Antonio Moreno Ibarra y a los profesores que me impartieron clases, gracias por compartirme su conocimiento.

Al Centro de Investigación en Computación (CIC) por permitirme formar parte de este gran centro de investigación.

Al Consejo Nacional de Ciencia y Tecnología (CONACyT) y al Programa Institucional de Formación de Investigadores (PIFI) y al Instituto Politécnico Nacional por brindarme apoyo económico.

No puedo terminar sin agradecer a mi papá Adán Cruz, a mi mamá Victoria Trejo, a mis hermanos Arturo y Araceli por su apoyo, cariño, ejemplo y enseñanzas en cada etapa de mi vida y Hermain Pérez Estrada por su apoyo, por compartir conmigo su conocimiento y por toda su ayuda en esta etapa.

Gracias a todos por apoyarme y confiar en mi. Y sobre todo gracias a Dios.

Índice general

Índice de figuras

Índice de tablas

Introducción

Una red móvil ad hoc (MANET) se compone de dispositivos móviles que pueden comunicarse entre sí mediante canales inalámbricos, donde no existe una infraestructura física preestablecida y la administración se realiza de forma descentralizada.

En este contexto los nodos participan en la toma de decisiones para el mantenimiento de la red. En este tipo de redes todos los nodos realizan tareas equivalentes, por lo tanto, para cada una de sus funciones es necesario desarrollar algoritmos distribuidos.

Por otro lado, los recursos como ancho de banda, capacidad de las baterías, memoria y poder de cómputo suelen ser limitados en este tipo de redes, por lo que los algoritmos distribuidos que las soportan se deben diseñar para optimizar su uso. En este contexto aparecen problemas de optimización de complejidad elevada que requieren de soluciones distribuidas, lo que los hace particularmente desafiantes.

Entre los problemas de optimización que podemos encontrar está el cálculo de conjuntos dominantes [4, 8, 9, 28, 30], de árboles de Steiner [8, 14, 20, 23], coloreado de gráficas [3, 28] y conjuntos independientes [3, 19, 20]. Se sabe que estos problemas de optimización son del tipo NP-difícil [13], por lo que no es posible calcular soluciones exactas en tiempo polinomial a menos que P = NP. Más aún, la solución exacta a estos problemas requiere información completa de las instancias a resolver (por ejemplo, la topología completa de la gráfica), lo cual no es factible en el contexto de las MANETs.

Por lo anterior, es necesario desarrollar algoritmos que sean capaces de encontrar soluciones en tiempo polinomial y que no requieran conocimiento completo de la instancia a resolver, aunque el resultado que entreguen sólo sea una aproximación del óptimo.

1.1. Algoritmos de aproximación

Hacer una aproximación es realizar una estimación o cálculo para obtener un resultado tan

cercano como sea posible al exacto o real, con el fin de llevar a cabo un determinado propósito. El objetivo de un algoritmo de aproximación para un problema dado, es ofrecer una solución aproximada a la solución óptima del problema.

Hacemos uso de un algoritmo de aproximación cuando la solución óptima del problema es inalcanzable o muy costosa, refiriéndonos al tiempo o a algún otro recurso requerido para resolver el problema de manera computacional. En este caso es razonable sacrificar la optimalidad y conformarse con una buena solución que pueda ser calculada eficientemente. Por supuesto, nos gustaría sacrificar tan poca optimalidad como sea posible, mientras ganamos tanta eficiencia como sea posible. Intercambiar optimalidad por tratabilidad es el paradigma de los algoritmos de aproximación.

Lo más importante de los algoritmos de aproximación es que garantizan encontrar una solución aproximada con un parámetro ρ que indica, para el peor de los casos, qué tan lejos de la solución óptima están las soluciones encontradas por un algoritmo de aproximación. Lo más interesante es que se puede demostrar que un algoritmo de aproximación siempre obtendrá un valor cercano al óptimo aún sin saber la solución óptima del problema.

Esta tesis tiene como uno de sus principales objetivos el estudio de los algoritmos de aproximación, enfocándonos en problemas que se presentan en el contexto de los sistemas distribuidos.

1.2. Antecedentes

Existe una gran cantidad de problemas computacionales importantes que ocurren en la vida real, muchos de ellos intratables, es decir, problemas para los que no se conoce un algoritmo eficiente o rápido para resolverlos. En teoría de la complejidad a esta clase de problemas se le conoce como NP-difícil.

Ya que no se les conoce soluciones eficientes a los problemas NP-difíciles para cualquier instancia de entrada, se han propuesto diversos métodos para atacarlos. Entre ellos están: las heurísticas, la programación dinámica, la búsqueda local, y los algoritmos de aproximación.

Garey et al. [12] y más tarde Johnson [18] formalizaron el concepto de algoritmo de aproximación. Un algoritmo de aproximación es necesariamente polinomial, y se evalúa en el

error relativo al peor de los casos entre todas las instancias del problema. Un algoritmo *A* es un algoritmo ρ-aproximado para un problema de minimización Π si para cada instancia I de Π, el algoritmo entrega una solución que tiene un valor a lo más ρ veces el óptimo. Naturalmente $\rho > 1$ y es mejor cuanto más se aproxima a 1. Similarmente para problemas de maximización un algoritmo ρ-aproximado entrega para cada instancia I una solución que es al menos ρ veces el óptimo. En este caso $\rho < 1$. A ρ se le conoce como factor de aproximación. Para el problema de maximización es también común referirse a $1/\rho$ como el factor de aproximación.

Cualquier heurística puede convertirse en un algoritmo de aproximación si para cualquier instancia del problema se demuestra analíticamente la aproximación con respecto al óptimo que tienen sus respectivas soluciones. Conocer esta aproximación es de gran ventaja, ya que proporciona la garantía implícita de que el algoritmo no entregará soluciones arbitrariamente lejanas al óptimo. Además, a partir de estos algoritmos se pueden proponer heurísticas con mejor calidad de solución, o en otras palabras, heurísticas con mejor factor de aproximación.

1.3. Planteamiento del problema

En el contexto de las MANETs es común encontrar diversos problemas de optimización debido a que para este tipo de redes es crucial hacer un uso eficiente de los recursos limitados de los que dispone. Por ejemplo, para evitar interferencia y utilizar el número mínimo de bandas de frecuencia, es necesario resolver problemas de coloreado de gráficas para encontrar árboles de distribución eficientes o resolver problemas de construcción de árboles de Steiner de peso mínimo, etc. Estos problemas tienen la particularidad de ser NP-difíciles.

Una de las operaciones más comunes en MANETs y en general en los sistemas distribuidos es la diseminación de información a todos los nodos de la red. Para tratar este problema se emplean diferentes algoritmos. Uno de los algoritmos más simples es la técnica de inundación (*flooding*).

En la técnica de inundación cuando un nodo recibe un paquete, este nodo lo retransmite a todos sus vecinos. Esto continúa hasta que todos los nodos de la red reciben el paquete. Sin embargo este mecanismo tiene varias deficiencias, por ejemplo, el mismo paquete puede ser enviado múltiples veces a un mismo nodo, no se toma en cuenta la energía disponible, etc.

Estas deficiencias limitan la escalabilidad debido al consumo de demasiados recursos como son: ancho de banda, energía y espacio en colas.

Como en el caso de la asignación de frecuencias y de la construcción de árboles de distribución, el problema de la diseminación de información puede ser modelado como un problema NP-difícil en el que se busca encontrar el subconjunto mínimo de nodos tal que si esos nodos retransmiten la información ésta llegará a todos los nodos de la red con el consecuente ahorro en el número de transmisiones.

Para abordar el problema de manera matemática una red ad hoc homogénea en dos dimensiones se modela frecuentemente como una gráfica de discos unitarios o UDG (*Unit Disk Graph*), y el problema NP-difícil a resolver se abstrae como el problema del conjunto dominante conectado mínimo en UDGs. Más recientemente se está utilizando otro tipo de gráficas, las gráficas de esferas unitarias o UBG (*Unit Ball Graph*), estas gráficas modelan una red ad hoc homogénea en tres dimensiones, lo que permite abstraer la red con más precisión. Bajo este modelo, el problema a resolver es el problema del conjunto dominante conectado mínimo en UBGs en ambientes distribuidos.

El conjunto dominante de un gráfica $G=(V,E)$ es el subconjunto mínimo V' de V tal que cada vértice que no pertenezca a V' está unido a al menos un miembro de V' por una arista $e \in E$. En este sentido si calculamos el conjunto dominante mínimo de la red habremos encontrado al conjunto mínimo de nodos que necesitan retransmitir un mensaje para que sea recibido por todos los nodos de la red. Sin embargo, para resolver el problema de la diseminación de información es necesario que el subconjunto de retransmisores sea conexo, es por este motivo que el problema NP-difícil a resolver es el problema de los conjuntos dominantes conectados.

Dado que el problema de los conjuntos dominantes conectados es NP-difícil [13] se deben proponer aproximaciones que nos entreguen soluciones en tiempo polinomial, y dado que son MANETs las soluciones también deben ser distribuidas y no pueden asumir que se conoce la topología completa de la red.

En esta tesis se busca caracterizar de forma experimental la efectividad de los algoritmos de aproximación para el cálculo de conjuntos dominantes que resuelvan el problema de la diseminación de información en una MANET.

La caracterización de los algoritmos se hará en términos de las métricas de desempeño de un algoritmo de diseminación de información, que son: relación de entrega de paquetes (*delivery ratio*), número total de paquetes transmitidos (*total overhead*) y retardo promedio de extremo a extremo del paquete (*end-to-end delay*). Adicionalmente se caracterizará de manera experimental el factor de aproximación de los diferentes algoritmos. Para esto se debe desarrollar un algoritmo que calcule al conjunto dominante conexo de la red. Asumiremos que este algoritmo posee información perfecta acerca de la topología de la red, la cual, como hemos mencionado, no está al alcance de los algoritmos distribuidos.

En la presente tesis se propone un nuevo algoritmo distribuido para el cálculo de conjuntos dominantes conexos que sea capaz de mejorar el desempeño, en términos las métricas antes mencionadas, de los algoritmos propuestos hasta hoy. El algoritmo propuesto está diseñado para trabajar en el contexto de las MANETs con modelos de propagación en tres dimensiones.

1.4. Objetivos

El objetivo principal de esta tesis es desarrollar y caracterizar experimentalmente un algoritmo de aproximación distribuido para el cálculo de conjuntos dominantes conectados en el contexto de las MANETs con modelos de propagación de señales electromagnéticas en tres dimensiones.

Los objetivos particulares son:

- Realizar un estudio teórico de algoritmos distribuidos de aproximación que calculen conjuntos dominantes conexos.
- Implementar en el entorno NS2 los principales algoritmos de aproximación distribuidos de aproximación que calculen conjuntos dominantes conexos.
- Desarrollar un nuevo algoritmo distribuido para el cálculo de conjuntos dominantes conexos.
- Realizar análisis experimentales para caracterizar el desempeño de los algoritmos.

1.5. Justificación

Es muy importante dar soluciones en tiempo polinomial a problemas NP-difíciles. Para darnos una idea de esta importancia, observemos la drástica diferencia entre la tasa de crecimiento del tiempo de ejecución de un algoritmo polinomial típico tal como n^3 y uno exponencial tal como 2^n. Por ejemplo, para n igual a 1000, n^3 es igual a mil millones, que es un número grande pero manejable, mientras 2^n es un número mucho más grande que el número de átomos en el universo. Los algoritmos de tiempo polinomial son lo suficientemente rápidos para muchos propósitos, pero los algoritmos de tiempo exponencial raramente son útiles.

Los algoritmos de aproximación nos ofrecen soluciones en tiempo polinomial y nos proporcionan mejores resultados de manera experimental. Además, nos proporcionan ideas para realizar heurísticas con mayor eficacia, lo que implica soluciones más cercanas a la solución óptima. Por otro lado, la métrica ρ de un algoritmo de aproximación nos garantiza que no encontraremos una solución arbitrariamente lejana al óptimo.

El uso de los algoritmos de aproximación para dar soluciones al problema de diseminación de información en el contexto de las MANETs es importante, ya que nos permiten gastar menos recursos haciendo posible una mejora en términos de la escalabilidad, que es uno de los problemas más fuertes en las MANETs [21].

1.6. Organización de la tesis

El resto de la presente tesis está organizado de la siguiente manera. En el Capítulo 2, titulado “Tratabilidad computacional”, se aborda la teoría de completitud haciendo énfasis en los conceptos y técnicas relacionados con los problemas prácticos de la tesis. En el Capítulo 3, titulado “Algoritmos de aproximación para conjuntos dominantes conectados”, se aborda la teoría de los algoritmos de aproximación y se analiza el funcionamiento de algunos de los algoritmos de este tipo, que construyen conjuntos dominantes conexos en diferentes tipos de gráficas. En el Capítulo 4, titulado “Aproximaciones distribuidas”, se analiza una serie de algoritmos utilizados para calcular conjuntos dominantes conexos pero ahora de manera

distribuida. En el Capítulo 5, titulado “Implementación de los algoritmos”, se da un breve introducción del entorno de simulación utilizado para implementar los algoritmos y se presenta el algoritmo propuesto en esta tesis. En el Capítulo 6, titulado “Análisis experimental”, se describen la métricas utilizadas para realizar la evaluación de los algoritmos, los escenarios que se utilizan en las simulaciones y se presentan los resultados obtenidos de manera gráfica. Finalmente, en el Capítulo 7 se presentan las conclusiones y aportaciones derivadas de esta tesis.

Resumen

Una red móvil ad hoc (MANET) está conformada por dispositivos móviles (a los que llamaremos nodos) que pueden comunicarse entre sí mediante canales inalámbricos donde no existe una infraestructura física preestablecida y la administración se realiza de forma descentralizada.

Una de las operaciones más comunes en una MANET es la *diseminación de información* a todos los nodos de la red. Para tratar este problema se emplean diferentes algoritmos entre los cuales está la técnica de *inundación*, donde cada que un nodo recibe un paquete por primera vez, lo retransmite a todos sus vecinos. Este mecanismo tiene varias deficiencias que limitan la escalabilidad debido al consumo excesivo de recursos tales como ancho de banda, energía y espacio en colas.

El problema de optimización relacionado con la diseminación de información tiene por objetivo reducir el número de transmisiones que se hacen para entregar un paquete a todos los nodos de la red. Para lograr esta tarea es necesario encontrar un *conjunto dominante conectado* de la red, donde únicamente los nodos que forman parte del conjunto dominante conectado son encargados de retransmitir los paquetes de diseminación. Es bien conocido que el problema del conjunto dominante conectado es NP-difícil.

En esta tesis se propone un nuevo algoritmo de aproximación distribuido para el calculo de conjuntos dominantes conectados bajo el modelo de gráficas de esferas unitarias. Por medio de un análisis experimental basado en simulaciones realistas, comparamos el desempeño del *algoritmo propuesto* contra el de los algoritmos de aproximación existentes para el calculo de conjuntos dominantes en gráficas de esferas unitarias, así como con el desempeño del algoritmo de inundación. Los resultados de las simulaciones muestran que, en general, el algoritmo propuesto tiene un desempeño superior.

Abstract

A mobile ad hoc network (MANET) is composed of mobile devices (which we call nodes) that communicate with each other through wireless channels. In this type of networks, there is no pre-established physical infrastructure and the network management is fully distributed.

One of the most common operations in a MANET is *to disseminate information* along the whole network. One of the simplest algorithms to solve the dissemination problem is *flooding*, a technique where each time a node receives a packet for the first time, it forwards the packet to its neighbors. The main disadvantage of flooding is that it consumes too much network resources such as bandwidth, power and space in queues. The latter severely limits the network scalability.

The optimization version of the efficient information dissemination problem aims to reduce the number of transmissions needed to deliver a packet to all nodes in the network. To accomplish this task, it is necessary to calculate a *connected dominating set* of the network, where only the nodes that are part of the connected dominating set are responsible for forwarding the packets. It is well known that the connected dominating set problem is NP-hard.

In this thesis we propose a new distributed approximation algorithm for computing connected dominating set over unit ball graphs. Using a detailed simulation based experimental analysis, we compare the performance of our proposed algorithm against that of the algorithms that compose the state of the art in computing connected dominating set over unit ball graphs as well as against that of flooding. Our simulations results show that, in general, our algorithm outperforms the other solutions.

Intratabilidad computacional

Una de las preguntas más importantes en la teoría de computación es si P es igual a NP (términos que definiremos más adelante). Se ha demostrado que existen problemas *intratables* en NP representativos de toda esta clase, de tal forma que si estos problemas tienen algoritmos de tiempo polinómico entonces $P=NP$ y si no los tienen entonces $P \neq NP$. A estos problemas se les conoce como *NP-completos*.

La teoría que trata este tipo de problemas nos ayuda a probar si un nuevo problema es *tan difícil* como los reconocidos ya como NP-completos. Saber si un problema es NP-completo nos ayuda a saber qué enfoque nos ofrece una mayor posibilidad de obtener algoritmos eficientes para solucionar el problema. Ejemplos de los enfoques que podemos utilizar, son:

- Buscar algoritmos eficientes que resuelvan varios casos especiales del problema general, aunque no resuelvan eficientemente el caso general.
- Utilizar técnicas como búsqueda con retroceso o ramificación y poda, aunque en el peor de los casos la complejidad no sea polinómica.
- Desarrollar un algoritmo aproximado. Esto implica relajar el problema en busca de un algoritmo rápido que no garantiza devolver la solución óptima pero sí alguna razonablemente cerca de serlo.

2.1. Complejidad computacional

Un *problema* se describe dando: (1) una descripción general de todos sus parámetros, y (2) una sentencia que nos dice qué propiedades debe satisfacer la respuesta o solución. Para obtener una *instancia* del problema debemos especificar los valores particulares para todos los parámetros del problema

Por ejemplo, consideremos el problema del *agente viajero*. Sus parámetros consisten de un

conjunto finito $C=\{c_1, c_2, \ldots, c_m\}$ de ciudades y, para cada par de ciudades c_i , c_j en C , la distancia $d(c_i, c_j)$ entre ellas. Una solución es un ordenamiento ($C_{\Pi(1)}$, $C_{\Pi(2)}$, ..., $C_{\Pi(m)}$) de las ciudades, que minimicen la suma

$$\left[\sum_{i=1}^{m-1} d\left(c_{\pi(i)}, c_{\pi(i+1)}\right)\right] + d\left(c_{\pi(m)}, c_{\pi(1)}\right).$$

Esta expresión nos da la longitud del camino que comienza en $c_{\Pi(1)}$, visita a cada ciudad en la secuencia y regresa directamente a $c_{\Pi(1)}$ desde la última ciudad $c_{\Pi(m)}$.

Un algoritmo *resuelve* un problema Π si ese algoritmo se puede aplicar a toda instancia I de Π y éste garantiza obtener siempre una solución para la instancia I .

En general, nos interesa encontrar el algoritmo más *eficiente* para resolver un problema, tomando en cuenta todos los recursos necesarios para la ejecución del algoritmo. Sin embargo, normalmente nos referimos con eficiente al algoritmo más rápido, debido a que en la práctica los requisitos de tiempo son a menudo un factor dominante para saber si usamos o no un algoritmo.

Los requisitos de tiempo de un algoritmo se expresan en términos del *tamaño* de una instancia del problema. A menudo el tamaño se mide de manera informal, por ejemplo, para el problema de agente viajero, se utiliza comúnmente el número de ciudades para este propósito. Sin embargo, una instancia del problema incluye más datos de entrada, como son: las etiquetas de las ciudades, las distancias entre ciudades, etc.

Cada problema tiene asociado un *sistema de codificación*, que mapea instancias de un problema en cadenas que las describen, tomando en cuenta todos los datos de entrada. Entonces, la *longitud de entrada* para una instancia I de un problema Π se define como el número de símbolos en la descripción de I , obtenida por un sistema de codificación e para Π . La longitud de entrada es la que se utiliza como medida formal del tamaño de la instancia.

La *función de complejidad en el tiempo* de un algoritmo nos dice el tiempo máximo requerido para resolver una instancia del problema con un tamaño de entrada determinado. Esta función queda bien definida hasta que se fija el sistema de codificación que se utilizará para

determinar la longitud de entrada y la computadora o modelo de computación que se utilizará para determinar el tiempo de ejecución. Sin embargo, estas decisiones particulares tienen poco efecto en la teoría de NP-completitud [13].

2.2 Algoritmos de tiempo polinómico y problemas intratables

Los algoritmos son divididos principalmente en dos tipos: los algoritmos de tiempo polinómico y los algoritmos de tiempo exponencial.

Decimos que una función $f(n) \in O(g(n))$ si existe una constante c tal que $|f(n)| \leq c|g(n)|$ para todos los valores de $n \geq n_0$. Un *algoritmo de tiempo polinómico* es aquel que tiene una función de complejidad del tiempo igual a $O(p(n))$ para algún polinomio p , donde n denota el tamaño de la entrada. Cualquier otro algoritmo cuya función de complejidad del tiempo no se pueda limitar de tal forma se llama *algoritmo de tiempo exponencial*, esta definición incluye ciertas funciones de complejidad no polinómica, como $n^{\log n}$, que normalmente no se consideran funciones exponenciales.

En la Tabla 2.1 se muestran las diferencias entre las tasas de crecimiento para algunas funciones de complejidad típica para los dos tipos de algoritmos.

Función de complejidad en el tiempo	**Tamaño de n**					
	10	20	30	40	50	60
n	0.00001s	0.00002s	0.00003s	0.00004s	0.00005s	0.00006s
n^2	0.0001s	0.0004s	0.0009s	0.0016s	0.0025s	0.0036s
n^3	0.001s	0.008s	0.027s	0.064s	0.125s	0.216s
n^5	0.1s	3.2s	24.3s	1.7m	5.2m	13.0m
2^n	0.001s	1.0s	17.9m	12.7 días	35.7 años	366 siglos
3^n	0.059s	58m	6.5 años	3855 siglos	2×10^8 siglos	1.3×10^{13} siglos

Tabla 2.1: Comparativo entres algunas funciones de complejidad polinómica y complejidad exponencial.

La tabla indica algunos de los motivos del porque los algoritmos de tiempo polinómico se consideran mucho mejores que los algoritmos de tiempo exponencial. Como podemos observar, también se tienen excepciones para instancias de cierto tamaño. Por ejemplo, para $n \leq 20$ el algoritmo con función exponencial 2^n es mucho más rápido que el algoritmo con

función polinómica n^5 .

Para la teoría de NP-completitud y la intratabilidad inherente, es importante tener clara la diferencia entre los algoritmos de tiempo polinómico y los algoritmos de tiempo exponencial, además de que se considera mejor a los primeros.

Se considera que un problema no ha sido bien resuelto hasta que se conoce un algoritmo de tiempo polinómico que lo resuelve. Por tanto, se dice que un problema es *intratable* si es tan difícil que no existe un algoritmo de tiempo polinómico que lo resuelve.

2.3. NP-completitud

La teoría de NP-completitud se aplica sólo a *problemas de decisión*. Un problema de decisión Π consta de un conjunto de *instancias* D_Π y un subconjunto de *sí-instancias* $Y_\Pi \subseteq D_\Pi$. La descripción de este tipo de problemas consiste de una *instancia del problema* y de una *pregunta*, cuya respuesta es *sí* o *no*, en términos de la instancia.

La restricción a problemas de decisión es su adecuado lenguaje para su estudio de manera matemática. Sea un conjunto finito de símbolos Σ y Σ^* el conjunto de todas las cadenas finitas de símbolos de Σ . Se dice que L es un *lenguaje* sobre el alfabeto Σ si L es un subconjunto sobre Σ^* .

La correspondencia entre los problemas de decisión y los lenguajes se realiza mediante los sistemas de codificación usados para especificar las instancias del problema. Por tanto, el problema Π y el sistema de codificación e dividen Σ^* en tres clases de cadenas: las que no son codificaciones de instancias de Π , las que codifican instancias de Π para las cuales la respuesta es *no* y las que codifican instancias de Π para las cuales la respuesta es *sí*. Esta tercer clase de cadenas es el lenguaje que asociamos con Π y e estableciendo de esta manera que:

$$L[\Pi, e] = \left\{ x \in \Sigma^* : \begin{array}{c} \Sigma \text{ es el alfabeto utilizado por } e \text{ y } x \text{ es la} \\ \text{codificación bajo } e \text{ de una instancia } I \in Y_\Pi \end{array} \right\}$$

La teoría se aplica a problemas de decisión diciendo que, si un resultado es válido para el

lenguaje $L[\Pi,e]$, entonces es válido para el problema bajo el esquema de codificación e.

2.3.1. Computación determinista y la clase P

Para formalizar el concepto de algoritmo se requiere un modelo de computación. Para este fin, haremos uso de la Máquina de Turing Determinista de una sola cinta o MTD, la cuál consiste de un control de estados finito, una cabeza de lectura y escritura y una cinta compuesta por una secuencia infinita de cuadros de cinta etiquetados como $\ldots,-2,-1,0,1,2,\ldots$ como se muestra en la Figura 2.1.

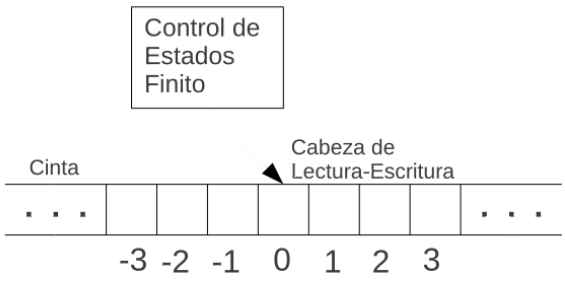

Figura 2.1: Representación de una Máquina de Turing Determinista (MTD) de una sola cinta.

Un *programa* para una MTD especifica la siguiente información:

1. Un conjunto finito Γ de símbolos de la cinta, incluyendo un subconjunto $\Sigma \subset \Gamma$ de símbolos de entrada y un *símbolo llamado blanco* $b \in \Gamma - \Sigma$.
2. Un conjunto finito de *estados* Q, que incluye un *estado de inicio* q_0 y dos *estados finales* q_Y y q_N.
3. Una función de transición $\delta:(Q-\{q_Y, qN\})\times\Gamma \rightarrow Q\times\Gamma\times\{-1,+1\}$.

La entrada a la MTD es una cadena $x \in \Sigma^*$. La cadena x se coloca sobre la cinta del cuadro 1 hasta el cuadro $|x|$, un símbolo por cuadro. Los demás cuadros se inician con el símbolo blanco. El programa inicia su operación en el estado q_0 con la cabeza de lectura y escritura revisando el cuadro 1. El cálculo procede paso a paso. Si el estado actual q es q_Y o q_N entonces el cálculo ha terminado con la respuesta *sí* si $q=q_Y$ y *no* si $q=q_N$. De lo contrario

el estado actual q pertenece a $Q-\{q_Y, q_N\}$, el símbolo $s \in \Gamma$ en la cinta es el símbolo que está siendo revisado, y además tenemos el valor $\delta(q,s)$ definido. Supongamos que $\delta(q,s)=(q',s',\delta)$. La cabeza de lectura y escritura entonces, borra s , escribe s' en su lugar, y se mueve un cuadro a la izquierda si $\delta=-1$, o un cuadrado a la derecha si $\delta=+1$. Al mismo tiempo, el control de estado finito cambia su estado de q a q' . Esto completa un *paso* de la computación, y se procede al siguiente paso, si es que lo hay.

En general, podemos decir que un programa MTD *M* con el alfabeto de entrada Σ acepta $x \in \Sigma^*$ si *M* se detiene en el estado q_Y cuando se aplica a la entrada x .

El reconocimiento del lenguaje no exige que *M* se detenga para *todas* las cadenas de entrada en Σ^* , sólo para aquellas en L_M . Si x pertenece a $\Sigma^* - L_M$, entonces el cálculo de *M* sobre *x* podría detenerse en el estado q_N , o podría continuar para siempre sin parar. Sin embargo, para un programa MTD que corresponda al concepto de *algoritmo*, se debe detener para todas las cadenas posibles del alfabeto de entrada que sean codificaciones de instancias de Π .

Se dice que un programa de MTD *M resuelve* el problema de decisión Π bajo el sistema de codificación e si *M* se detiene para todas las cadenas de entradas del alfabeto que sean codificaciones de instancias de Π y $L_M = L[\Pi, e]$.

Un programa MTD también calcula funciones. Por ejemplo, si *M* es un programa MTD con el alfabeto de entrada Σ y el alfabeto de cinta Γ que se detiene para todas las cadenas de entrada Σ^* . Entonces *M* calcula la *función* $f_M : \Sigma^* \to \Gamma^*$ donde, para cada $x \in \Sigma^*$, $f_M(x)$ es la cadena obtenida al ejecutar *M* sobre la entrada x hasta que se detiene formando una cadena de símbolos sobre las casillas de la cinta.

El modelo MTD nos proporciona una equivalencia formal de un algoritmo en al cual se basan algunas de las definiciones posteriores.

El *tiempo* utilizado en el cálculo de un programa MTD *M* para la entrada *x* es el número de pasos que ocurren en el cálculo hasta que se presenta un estado final. Para un programa MTD *M* que se detiene para todas las entradas $x \in \Sigma^*$ que sean codificaciones de instancias de Π ,

su función de complejidad en el tiempo $T_M : Z^+ \to Z^+$ esta dada por

$$T_M(n) = max \left\{ m : \begin{array}{c} \text{existe } x \in \Sigma^*, \text{ con } |x| = n \text{ tal que el cálculo} \\ \text{de } M \text{ para la entrada } x \text{ toma tiempo } m \end{array} \right\}$$

El programa *M* se llama *programa MTD de tiempo polinómico* si existe un polinomio *p* tal que, para todo $n \in Z^+, T_M(n) \leq p(n)$.

La definición formal de la clase P se define de la siguiente manera:

Definición 1 P = { L : *existe un programa MTD de tiempo polinómico para el cuál* $L = L_M$ }

Un problema de decisión pertenece a *P* bajo el sistema de codificación e si $L[\Gamma, e] \in P$, es decir, si existe un programa MTD de tiempo polinómico que *resuelve* Π bajo el sistema de codificación e . Puesto que ya se mencionó la equivalencia entre los sistemas de codificación, por lo general se omite la especificación del sistema de codificación particular, diciendo que el problema de decisión Π pertenece a *P*.

También se utiliza de manera informal el término *algoritmo de tiempo polinómico*, la parte formal es el programa MTD de tiempo polinómico. Sin embargo, la definición formal de *P* se puede formular en términos de programas para cualquier modelo computacional y resulta la misma clase de lenguajes. De hecho, los algoritmos se tratan de una manera casi independiente del modelo, refiriéndose a cómo operan directamente sobre los componentes de una instancia y no en sus descripciones codificadas.

Computación no determinista y la clase NP

Existen problemas para los cuales no se conoce algún algoritmo de tiempo polinómico que los resuelva. Sin embargo para una instancia particular de un problema de decisión, alguien puede decir que la respuesta es *sí* y se le puede pedir que demuestre tal afirmación. El *procedimiento de verificación* correspondiente se puede especificar como un algoritmo que pudiera tener complejidad de tiempo polinómico.

La verificación en tiempo polinómico no implica que el problema se resuelva en tiempo

polinómico. Al decir que se puede verificar una respuesta *sí* para una instancia; no se cuenta el tiempo que se puede llevar buscar, entre la cantidad exponencial de soluciones posibles, una que cumpla los requerimientos deseados.

Informalmente NP se define en términos de lo que se llama un *algoritmo no determinista*. Este tipo de algoritmos se compone de dos etapas, la primera es la *etapa de adivinante* y la segunda, es la *etapa de verificación*. Dada una instancia I de un problema, la primera etapa sólo adivina alguna estructura o posible solución S. A continuación, se proporciona I y S como entradas a la etapa de verificación, que calcula de manera determinista normal, si finalmente se detiene con la respuesta *sí*, o con la respuesta *no*, o si continua calculando sin detenerse.

Un algoritmo no determinista resuelve un problema de decisión Π, si para todas las instancias $I \in D_{\Pi}$ se cumplen las siguientes dos propiedades:

1. Si $I \in Y_{\Pi}$, entonces existe una estructura S que cuando es adivinada para la entrada I, la etapa de verificación responde *sí* para I y S.
2. Si $I \notin Y_{\Pi}$, entonces no existe una estructura S que cuando es adivinada para la entrada I, la etapa de verificación responde *sí* para I y S.

Un algoritmo no determinista que resuelve un problema de decisión Π se ejecuta en tiempo polinómico si existe un polinomio p tal que, para cada instancia $I \in Y_{\Pi}$, existe un estructura adivinada S y la etapa de verificación determinista responde *sí* para I y S en tiempo p.

La clase NP se define de manera informal como la clase de todos los problemas de decisión Π que, bajo sistemas de codificación, se pueden resolver por algoritmos no deterministas de tiempo polinómico.

La definición formal en términos de lenguajes y máquinas de Turing para un algoritmo no determinista es un programa para una máquina de Turing no determinista de una cinta (MTND).

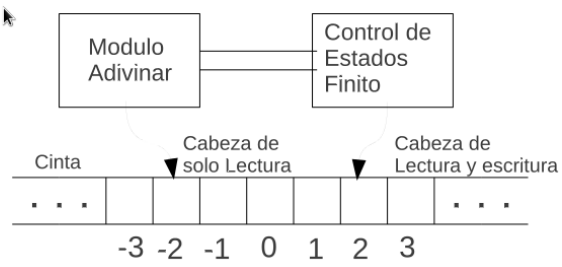

Figura 2.2: Representación de una máquina de Turing no determinista de una cinta (MTND).

El modelo MTND tiene la misma estructura que una MTD, pero además cuenta con un *módulo de adivinar* que tiene su propia *cabeza de sólo escritura*, como se muestra en la Figura 2.2. El módulo de adivinar proporciona los medios para escribir lo adivinado y se utiliza únicamente para este propósito.

Un programa MTND se especifica de la misma forma que un programa MTD, incluyendo un alfabeto de cinta Γ , un alfabeto de entrada Σ , un símbolo llamado blanco b , un conjunto de estados Q , un estado inicial q_0 , dos estados finales q_Y y q_N y una función de transición $\delta : (Q - \{q_Y, q_N\}) \times \Gamma \rightarrow Q \times \Gamma \times \{-1, +1\}$. El cálculo de un programa MTND sobre una cadena $x \in \Sigma^*$ se lleva a cabo en dos etapas, a diferencia de una MTD.

La primera etapa es la de *adivinar*. Inicialmente, la cadena de entrada x se escribe a partir del cuadro 1 hasta el cuadro $|x|$ de la cinta (mientras que todos los otros cuadros están en blanco), la cabeza de lectura-escritura revisa el cuadro 1 , la cabeza de sólo escritura revisa el cuadro -1 , y el control de estados finito está inactivo. El módulo de adivinar dirige la cabeza de sólo escritura, ya sea para escribir algún símbolo de Γ en el cuadro de la cinta que está siendo revisado, y mover un cuadro a la izquierda o dejar que, en ese momento el módulo de adivinar se vuelva inactivo y el control de estados finito se active en q_0 . La elección de si permanece activo, y si es así, qué símbolo de Γ escribir, lo hace el módulo de adivinar de forma arbitraria. Así, el módulo de adivinar puede escribir cualquier cadena de Γ^* antes de que se detenga.

La etapa de *verificación* comienza cuando el control de estados finito se activa en el estado q_0. A partir de este punto, el cálculo procede únicamente bajo la dirección del programa MTND de acuerdo con las mismas reglas que para una MTD. El módulo de adivinar y la cabeza de sólo

escritura ya no están involucrados, habiendo cumplido su papel al escribir la cadena adivinada en la cinta. Por supuesto, la cadena adivinada puede (y suele) ser examinada durante la fase de verificación. El cálculo termina cuando el control de estado finito entra en uno de los dos estados finales q_Y , o q_N . Se dice que *acepta* si se detiene en el estado q_Y . Todos los otros cálculos, se detengan o no, se clasifican como *no aceptados*.

Cualquier programa MTND M tendrá un número infinito de cálculos posibles para una cadena de entrada *x*, uno para cada posible cadena adivinada de Γ^* . Decimos que el programa de MTND *M acepta x* si al menos uno de los cálculos es aceptado. El lenguaje *reconocido* por *M* es:

$$L_M = x \in \Sigma : M \text{ acepta } x$$

El *tiempo* requerido para aceptar la cadena $x \in L_M$ por un programa MTND *M*, se define como el mínimo, de todos los cálculos de aceptación de *M* para x , número de pasos que ocurren en las etapas de adivinar y verificar hasta que el estado de alto q_Y se presenta. La *función de complejidad del tiempo* $T_M : Z^+ \rightarrow Z^+$ para *M* es

$$T_M(n) = max\left(\{1\} \cup \left\{m : \begin{array}{c}\text{existe una } x \in L_M, \text{ con } |x| = n \text{ tal que} \\ \text{el tiempo para aceptar } x \text{ por } M \text{ es } m\end{array}\right\}\right)$$

La función de complejidad del tiempo para *M* depende del número de pasos que ocurren en aceptar. Por convención $T_M(n)$ es igual a 1 cuando no hay entradas de longitud *n* aceptadas por *M*. El programa MTND *M* es un *programa MTND de tiempo polinómico* si existe un polinomio p tal que $T_M(n) \leq p(n)$ para todo $n \geq 1$ Por último, la clase NP se define formalmente como sigue:

Definición 2 *NP = {L: existe un programa MTND M de tiempo polinómico para el cuál* $L_M = L$*}*

Un problema de decisión Π pertenece a NP bajo el sistema de codificación *e*, si el lenguaje $L[\Pi, e] \in NP$. Al igual que con *P*, podemos decir que Π está en NP sin dar un sistema de codificación específico.

2.3.3 La relación entre P y NP

Como se vio, NP es la clase de lenguajes que se resuelven en tiempo polinómico sobre una MTND o bien, la clase de lenguajes donde sus miembros pueden ser verificados en tiempo polinómico y P es la clase de lenguajes donde los miembros se pueden decidir en tiempo polinómico. Lo anterior se puede resumir como:

- P = Es la clase de lenguajes que se pueden decidir rápidamente
- NP = Es la clase de lenguajes que se pueden verificar rápidamente

La primera observación es que $P \subseteq NP$. Es decir, todos los problemas de decisión resueltos por un algoritmo determinista de tiempo polinómico también pueden ser resueltos por un algoritmo no determinista de tiempo polinómico. Ya que cualquier algoritmo determinista puede ser usado como fase de verificación de un algoritmo no determinista. Por otro lado, nadie ha probado que exista un problema en NP que no esté en P. La pregunta $¿ P = NP ?$ es una de las más misteriosas e importantes en la teoría de la computación.

Para demostrar que $P \neq NP$ tendríamos que encontrar un problema en NP que no esté en P, mientras que para probar que $P = NP$, en principio tendríamos que encontrar un algoritmo de tiempo polinómico para cada problema en NP. Pero esta tarea se puede simplificar considerablemente, ya que es suficiente encontrar un algoritmo de tiempo polinómico para uno solo de los problemas de la clase NP-completo.

2.3.4 Transformaciones polinómicas y NP-completo

Supongamos que queremos resolver el problema de decisión Π_1 y que tenemos un algoritmo que resuelve el problema de decisión Π_2 y también que tenemos un algoritmo que construye una instancia y de Π_2 para toda instancia x de Π_1 , de tal forma que un algoritmo para Π_2 responde *sí* para y si y sólo si la respuesta al problema Π_1 para x es *sí*. Dicho algoritmo se denomina *algoritmo de transformación*. El algoritmo de transformación combinado con el algoritmo para Π_2 nos da un algoritmo para Π_1 , como se representa en la Figura 2.3.

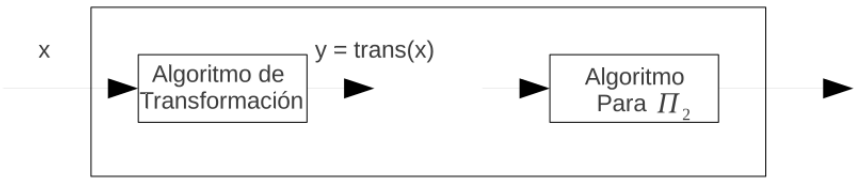

Figura 2.3: Algoritmo para Π_1.

Si existe una transformación polinómica del problema de decisión Π_1 en el problema de decisión Π_2, el problema Π_1 se transforma en tiempo polinómico al problema Π_2.

De manera formal, una *transformación polinómica* de un lenguaje $L_1 \subseteq \Sigma_1^*$, a un lenguaje $L_2 \subseteq \Sigma_2^*$ es una función $f : \Sigma_1^* \to \Sigma_2^*$ que satisface las siguientes dos condiciones:

1. Existe un programa MTD de tiempo polinómico que calcula f.
2. Para todo $x \in \Sigma_1^*$, $x \in L_1$ si y solo si $f(x) \in L_2$.

Si existe una transformación polinómica de L_1 a L_2, escribimos $L_1 \propto_p L_2$, que se lee L_1 se transforma polinómicamente en L_2.

La importancia de las transformaciones polinómicas viene del Teorema 1:

Teorema 1 Si $L_1 \propto_p L_2$, entonces $L_2 \in P$ implica $L_1 \in P$ (y, de manera equivalente $L_2 \notin P$, implica $L_1 \notin P$).

Si Π_1 y Π_2 son problemas de decisión, con sus correspondientes sistemas de codificación e_1 y e_2, escribimos $\Pi_1 \propto_p \Pi_2$ siempre que exista una transformación polinómica de $L[\Pi_1, e_1]$ a $L[\Pi_2, e_2]$. Se puede considerar una transformación polinómica del problema de decisión Π_1 al problema de decisión Π_2 como una función $f : D_{\Pi_1} \to D_{\Pi_2}$ que satisface las siguientes dos condiciones:

1. f es calculable por un algoritmo de tiempo polinómico, y
2. Para todo $I \in D_{\Pi_1}$, $I \in Y_{\Pi_1}$ si y solo si $f(I) \in Y_{\Pi_2}$

La relación *transformación polinómica* es útil ya que es transitiva, como se muestra en el

Teorema 2:

Teorema 2 Si $L_1 \propto_p L_2$ y $L_2 \propto_p L_3$, entonces $L_1 \propto_p L_3$.

Definición 3 *Un lenguaje es NP-completo si:*

1. $L \in NP$ *y*
2. *Para todos los demás lenguajes* $L' \in NP$, $L' \propto_p L$

De manera informal, un problema de decisión Π es NP-completo si $\Pi \in NP$ y, para todos los problemas de decisión $\Pi' \in NP$, $\Pi' \propto_p \Pi$. El Teorema 1, lleva a identificar los problemas NP-completos como *los problemas más difíciles de NP*, ya que si uno de los problemas NP-completo puede ser resuelto en tiempo polinómico, entonces *todos* los problemas en NP también pueden ser resueltos en tiempo polinómico. Si cualquier problema en NP es intratable, entonces también todos los problemas NP-completos son intratables. Por lo tanto, un problema NP-completo Π tiene la propiedad de que: si $P \neq NP$, entonces $\Pi \in NP - P$. Más precisamente, $\Pi \in P$ si y sólo si $P = NP$.

Teorema 3 Si L_1 y L_2 pertenecen a NP, L_1 es NP-completo y $L_1 \propto_p L_2$, entonces L_2 es NP-completo.

Demostración. Ya que $L_2 \in NP$, se necesita mostrar que para cada $L' \in NP$, $L' \propto_p L_2$. Considerando que cualquier $L' \in NP$. Ya que L_1 es NP-completo, debe ser el caso de que $L' \propto_p L_1$. La transitividad de $\propto$ y el hecho de que $L_1 \propto_p L_2$ entonces implica que $L' \propto_p L_2$.

Para problemas de decisión, este Teorema nos da un enfoque sencillo para probar nuevos problemas NP-completos, una vez que tengamos al menos un problema NP-completo. Para probar que Π es NP-completo, sólo se debe mostrar que:

1. $\Pi \in NP$ y
2. Algún problema NP-completo Π' se transforma a Π

2.3.5. Teorema de Cook

El primer problema que se demostró es NP-completo es un problema de decisión de la lógica

booleana, conocido como el *problema de satisfacibilidad (SAT)*. Los términos que se utilizan para describirlo se definen como sigue:

Sea $U=\{u_1, u_2, \ldots, u_m\}$ un conjunto de *variables* booleanas. Un *asignación verdadera* para U es una función $t: U \to T,F$. Si $t(u)=T$ decimos que u es *verdadera* sometida a *t*; si $t(u)=F$ decimos que u es *falsa*. Si u es una variable en U, entonces u y $\bar{u}$ son *literales* sobre U. La literal u es verdadera bajo t si y sólo si la variable u es verdadera bajo t, la literal $\bar{u}$ es verdadera si y sólo si la variable u es falsa.

Una *cláusula* sobre U es un conjunto de literales sobre U, tal como $\{u_1, \bar{u}_3, u_8\}$, que representa la disyunción de las literales y se *satisface* con una asignación verdadera si y sólo si al menos uno de sus miembros es verdadero bajo esa asignación. La cláusula anterior será satisfecha por t a menos que $t(u_1)=F$, $t(u_3)=T$ y $t(u_8)=F$. Una colección C de clausulas bajo U es *satisfacible* si y solo si existe alguna asignación de verdad para U que satisfaga simultáneamente todas las cláusulas en C. Tal asignación de verdad se llama una *asignación de verdad satisfactoria* para C. El problema de satisfacibilidad se especifica de la siguiente manera:

Problema de satisfacibilidad:

Instancia. Un conjunto U de variables y una colección C de cláusulas sobre U.

Pregunta. ¿Existe alguna asignación de verdad satisfactoria para C ?

Teorema 4 (Teorema de Cook) El problema de satisfacibilidad es NP-completo

Demostración. Para mostrar este Teorema se debe cumplir con los dos requisitos de la Definición 3. Es fácil ver que SAT esta en NP, ya que un algoritmo no determinista para él, sólo necesita adivinar una asignación verdadera para las variables dadas y verificar si la asignación satisface todas las cláusulas de la colección dada C. Es claro que esto se puede hacer en tiempo polinómico no determinista. Así, el primero de los dos requisitos para probar que es NP-completo se cumple.

Para el segundo requisito, demostrar que para todos los lenguajes $L \in NP$, $L \propto L_{SAT}$, regresamos a nivel de lenguaje, donde SAT se representa por el lenguaje $L_{SAT}=L[SAT,e]$ bajo un sistema de codificación e. Existen muchos y muy diversos lenguajes en NP así que no se

puede esperar una transformación para cada uno de ellos. Sin embargo, todos los lenguajes en NP se pueden describir de forma estándar, dando un programa MTND de tiempo polinómico que los reconozca. Lo que permite trabajar con un programa MTND general de tiempo polinómico y deducir una transformación general del lenguaje que reconoce a L_{SAT} . Cuando esta transformación se enfoca a un programa MTND M particular que reconoce el lenguaje L_M , nos da la transformación polinómica buscada de L_M a L_{SAT} . Así, en esencia, se da una demostración simultánea para todo $L \in NP$ que $L \propto L_{SAT}$. La demostración completa se puede encontrar en [13].

2.3.6 Problemas NP-completos

Una vez que se probó el primer problema NP-completo, el procedimiento para probar otros problemas NP-completos se simplificó. Ya que dado $\Pi \in NP$, solo se requiere mostrar que algún problema Π' , ya reconocido como NP-completo, se puede transformar a Π . De esta forma los pasos para mostrar que un problema es NP-completo son:

1. Mostrar que Π esta en NP.
2. Seleccionar un problema NP-completo Π' .
3. Construir una transformación f de Π' a Π .
4. Probar que f es un transformación de tiempo polinómico.

Cualquier problema conocido NP-completo sirve para probar un nuevo problema NP-completo, sin embargo los siguientes problemas han sido utilizados con mayor frecuencia: 3-satisfacibilidad (3SAT), emparejamiento en 3-dimensiones (E3D), cubierta por vértices (CV), clique máximo, circuito hamiltoniano (CH) y el problema de la partición.

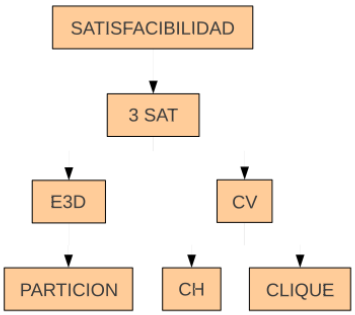

Figura 2.4: Diagrama de la secuencia usada para probar que los seis problemas básicos son NP-completos.

La Figura 2.4 muestra cual problema se transforma a cada uno de los seis problemas básicos, incluyendo el problema de cubierta por vértices, que es utilizado para demostrar otro problema NP-completo llamado el problema del conjunto dominante, ambos se especifican a continuación.

Problema de la cubierta de vértices (CV):

Instancia. Una gráfica $G=(V,E)$ y un entero positivo $k \leq |V|$

Pregunta. ¿Existe una *cubierta por vértices* de tamaño k o menos para G, o bien, existe un subconjunto $V' \subseteq V$ tal que $|V'| \leq k$ y para cada arista $(u,v) \in E$, al menos uno de los vértices u o v pertenece a V'?

Problema del conjunto dominante:

Instancia. Una gráfica $G=(V,E)$ y un entero positivo $k \leq |V|$

Pregunta. ¿Existe un conjunto dominante de tamaño k o menos para G, o bien, existe un subconjunto $V' \subseteq V$ con $|V'| \leq k$ donde para todo $u \in V - V'$ existe $v \in V'$ tal que $(u,v) \in E$?

Es claro que el problema del conjunto dominante está en NP. Dado un conjunto dominante, se puede verificar en tiempo polinómico si efectivamente es un conjunto dominante. Esto se puede hacer tomando cada vértice y comprobando si está en el conjunto dominante, o bien si una de sus aristas esta unida a un nodo del conjunto dominante.

A continuación, se selecciona el problema de la cubierta de vértices ya reconocido como NP-completo y se construye la transformación f del problema de la cubierta de vértices al

problema del conjunto dominante, *cubierta de vértices* $\propto_p$ *conjunto dominante*, como se muestra a continuación.

Dada una gráfica G , la función f construye una gráfica G' de la siguiente manera. G' contiene todas las aristas y vértices de G , además para cada arista $(u,v) \in G$, añadimos un nodo intermedio w y las aristas (u,w) y (w,v) en G' , como se muestra en la Figura 2.5. Ahora, demostramos que G tiene una cobertura de vértices de tamaño k si y solo si G' tiene un conjunto dominante del mismo tamaño.

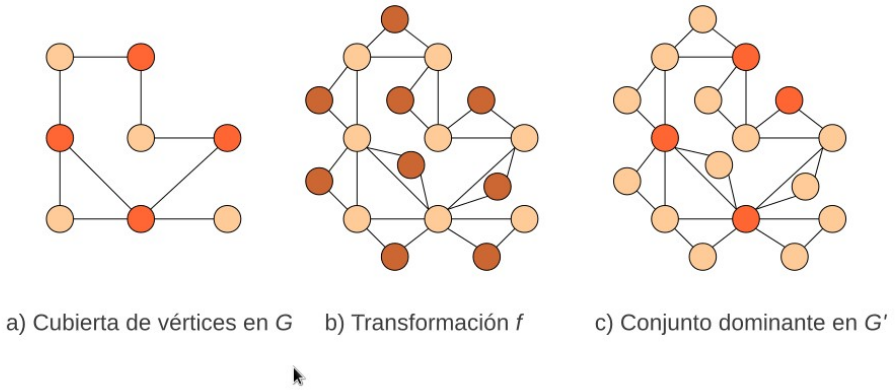

Figura 2.5: Cubierta de vértices de tamaño 4 en G, transformación f y conjunto dominante de tamaño 4 en G'.

Si S es una cubierta de vértices en G , se debe mostrar que S es un conjunto dominante de G' . S es una cubierta de vértices, esto significa que cada arista en G tiene algunos de sus vértices extremos en S . Consideremos $v \in G$, si v es un nodo original en G entonces o bien $v \in S$ o debe haber alguna arista conectando v con algún otro vértice u . Puesto que S es una cobertura de vértices, si $v \notin S$, entonces u debe estar en S y por tanto hay un vértice adyacente de v en S . Por lo tanto, v está dominado por algún elemento en S . Sin embargo, si w es un nodo adicional en G' , entonces w tiene dos vértices adyacentes $(u,v) \in G$ y con el argumento anterior al menos uno de ellos esta en S . Por lo tanto los nodos adicionales también están cubiertos por S . Así que si G tiene una cobertura de vértices, entonces G' tiene un conjunto dominante del mismo tamaño.

Ahora, si G' tiene un conjunto dominante D de tamaño k , entonces examinamos todos los vértices adicionales $w \in D$, cada uno de los cuales tiene exactamente dos vértices $(u,v) \in G$. Observamos que podemos reemplazar w por u o v . w en D nos ayuda a

dominar solo u , v , $w \in G'$, pero estas aristas forman un 3-ciclo y podemos elegir u o v y aún dominar todos los vértices que w solía dominar. Así que podemos eliminar todos los vértices adicionales. Y puesto que todos los vértices adicionales corresponden a una de las aristas en G y ya que todos los vértices adicionales están cubiertos por el modificado D , esto significa que todas las aristas en G están cubiertas por el conjunto. Así si G' tiene un conjunto dominante de tamaño k , entonces G tiene una cubierta de vértices de tamaño a los más k .

De esta manera, se demuestra ambos lados de la equivalencia. Un conjunto dominante de tamaño k existe en G' si y sólo si una cubierta de vértices de tamaño k existe en G . Puesto que sabemos que cubierta de vértices es un problema NP-completo, entonces el problema del conjunto dominante también es NP-completo.

2.4. NP-difícil

Cualquier problema de decisión Π , sea o no miembro de NP, al cual pueda transformarse un problema NP-completo, tendrá la propiedad de que no puede ser resuelto en tiempo polinómico a menos que $P=NP$. Ya que Π es al menos tan difícil como los problemas NP-completos, se dice que Π es *NP-difícil*, sin embargo el concepto NP-difícil es aún más general. Para explicarlo, se debe tomar en cuenta que es posible generalizar el concepto de transformación polinomial de tal forma que se puede demostrar que los problemas de búsqueda, además de los de decisión, son “al menos tan difíciles” como los problemas NP-completos.

2.4.1. Reducibilidad de Turing y problemas NP-Difíciles

Muchos de los problemas se presentan como *problemas de búsqueda*. Un problema de búsqueda Π consiste de un conjunto *instancias* D_Π y, para cada instancia $I \in D_\Pi$, un conjunto de *soluciones* $S_\Pi[I]$ para I . Se dice que un algoritmo *resuelve* un problema de búsqueda Π si, para cualquier instancia de entrada $I \in D_\Pi$, regresa como respuesta *no* cada vez que $S_\Pi[I]$ es vacío, y en caso contrario devuelve alguna solución s que pertenece a $S_\Pi[I]$.

Cualquier problema de decisión Π puede ser formulado como un problema de búsqueda

definiendo $S_{\Pi}[I]=sí$ si $I \in Y_{\Pi}$ y $S_{\Pi}[I]=\varphi$ si $I \notin Y_{\Pi}$. Por lo tanto, asumiendo que los problemas de decisión se formulan de esta manera, un problema de decisión puede ser considerado como un tipo especial de problema de búsqueda.

La parte formal de un problema de búsqueda es una *relación de cadena*. Para un alfabeto finito Σ, una relación de cadena sobre Σ es una relación binaria $R \subseteq \Sigma^{+} \times \Sigma^{+}$, donde $\Sigma^{+}=\Sigma^{*}-\{\varepsilon\}$, es decir, el conjunto de todas las cadenas no vacías sobre Σ. Un lenguaje L sobre Σ puede ser definido con la relación de cadena

$$R=\{(x, s): x \in \Sigma^{+} y x \in L\}$$

donde s es cualquier símbolo fijo de Σ. Una función $f: \Sigma^{*} \rightarrow \Sigma^{*}$ *genera* la relación de cadena R si y solo si, para cada $x \in \Sigma^{+}$, $f(x)=\varepsilon$ si no existe alguna $y \in \Sigma^{+}$ tal que $(x,y) \in R$ de otra forma $f(x)$ es igual a alguna $y \in \Sigma^{+}$ para la cual $(x,y) \in R$. Un programa MTD *M resuelve* la relación de cadena R si la función f_M calculada por *M* genera R.

La correspondencia entre los problemas de búsqueda y las relaciones de cadena se logra mediante los sistemas de codificación, sólo que ahora un sistema de codificación para Π debe dar una cadena codificada para cada instancia $I \in D_{\Pi}$ y una cadena codificada para cada solución $s \in S_{\Pi}[I]$. Bajo el sistema de codificación e, el problema de la búsqueda Π corresponde a la relación de cadena $R[\Pi,e]$ definida como:

$$R[\Pi, e]=\left\{(x, y): \begin{array}{c} x \in \Sigma^{+} \text{ es la codificación bajo } e \text{ de una instancia } I \in D_{\Pi} \text{ y} \\ y \in \Sigma^{+} \text{ es la codificación bajo } e \text{ de una solución } s \in S_{\Pi}[I] \end{array}\right\}$$

Se dice que Π se resuelve mediante un algoritmo de tiempo polinómico si existe un programa MTD de tiempo polinómico que resuelve $R[\Pi,e]$.

La generalización de “transformación polinómica” que se utiliza, procede del hecho de que una transformación polinómica de un problema de decisión Π a un problema de decisión Π' nos proporciona un algoritmo A para resolver Π mediante el uso de una “subrutina” hipotética para resolver Π'. Dada cualquier instancia I de Π, el algoritmo primero construye una instancia equivalente I' de Π', entonces aplica la subrutina para I ' y finalmente nos da la respuesta obtenida por la subrutina, que también es la respuesta correcta para I. El algoritmo

A se ejecuta en tiempo polinómico, excepto por el tiempo requerido por la subrutina. Por lo tanto, si la subrutina fuera un algoritmo de tiempo polinómico para resolver Π', entonces el procedimiento en general sería un algoritmo de tiempo polinómico para resolver Π.

Una *reducción de Turing de tiempo polinómico* de un problema de búsqueda Π a un problema de búsqueda Π' es un algoritmo *A* que resuelve Π usando una subrutina hipotética S para la resolver Π' de tal forma que si S fuera un algoritmo de tiempo polinómico para Π', entonces *A* sería un algoritmo de tiempo polinómico para Π.

La formalización del concepto anterior se realiza en términos de las llaman máquinas oráculo, como el modelo de Turing, aunque se puede utilizar cualquier modelo estándar de computación. Una *máquina de Turing oráculo (MTO)* consiste de una MTD estándar, aumentada con una *cinta oráculo* que tiene sus cuadros etiquetados como $\ldots,-2,-1,0,1,2,\ldots$ y una *cabeza oráculo* de lectura-escritura para trabajar con esta cinta. Esta máquina se muestra en la Figura 2.6.

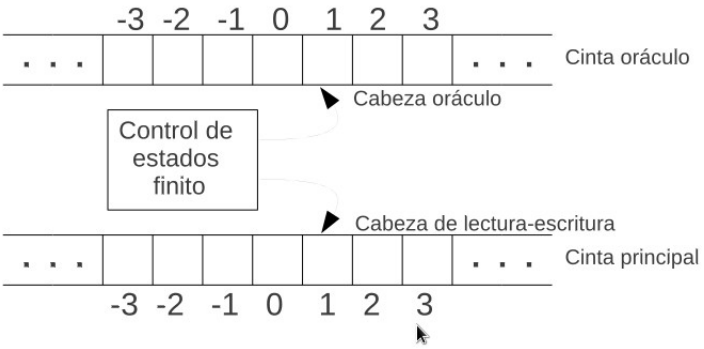

Figura 2.6: Representación de una Maquina de Turing Oráculo (MTO).

Un programa para una MTO es similar que el de una MTD y especifica lo siguiente:

1. Un conjunto finito Γ de *símbolos de cinta*, que incluye un subconjunto $\Sigma \subset \Gamma$ de *símbolos de entrada* y un *símbolo en blanco* $b \in \Gamma - \Sigma$.
2. Un conjunto finito *Q* de *estados*, incluyendo un *estado-inicial* q_0, un *estado-final* q_h, un *estado-oráculo de consulta* q_c, y un *estado de reanudación del cálculo* q_r.
3. Una función de transición $\delta : \left(Q - q_h, q_c\right) \times \Gamma \times \Sigma \to Q \times \Gamma \times \Sigma \times -1,+1 \times -1,+1$.

El cálculo de un programa MTO sobre una entrada $x \in \Sigma^*$ es similar que para una MTD excepto que, cuando el control de estados finito está en el estado q_c, lo que ocurre en el siguiente paso depende de una *función oráculo* especifica $g : \Sigma^* \to \Sigma^*$. EL cálculo comienza con los símbolos de x escritos del cuadro 1 al cuadro x de la cinta principal, el resto de los cuadros de esta cinta en blanco y todos los cuadros de la cinta oráculo, también en blanco. Además cada cabeza de lectura-escritura revisando el cuadrado 1 de su cinta y el control de estados finitos en el estado q_0. El cálculo avanza paso a paso con una de las siguiente tres posibilidades ocurriendo en cada paso:

1. Si el estado actual es q_h, entonces el cálculo termina y ningún otro paso se lleva a cabo.
2. Si el estado actual es $q \in Q - q_h, q_c$, entonces la acción que se toma dependerá de los símbolos que se estén revisando en las dos cintas y la función de transición δ. Sea s_1 el símbolo que está revisando la cabeza de la cinta principal y s_2 el símbolo que esta revisando la cabeza de la cinta oráculo y sea $(q', s_1', s_2', \Delta_1, \Delta_2)$ el valor de $\delta(q, s_1, s_2)$. El control de estados finito cambia del estado q al estado q', la cabeza de la cinta principal escribe s_1' en el lugar de s_1 y cambia su posición según Δ_1 (hacia adelante si $\Delta_1 = +1$ y hacia atrás si $\Delta_1 = -1$) y la cabeza oráculo escribe s_2' en el lugar de s_2 y cambia su posición según Δ_2. Por lo tanto, esto es como un paso de una MTD, excepto que involucra dos cintas.
3. Si el estado actual es q_c, entonces la acción tomada depende del contenido de la cinta oráculo y de la función oráculo g. Sea $y \in \Sigma^*$ la cadena que parece en los cuadros 1 a $|y|$ de la cinta oráculo, donde el cuadro $|y|+1$ es el primer cuadro a la derecha del cuadro 0 que contiene un símbolo blanco, y sea $z \in \Sigma^*$ el valor de $g(y)$. Entonces en un paso la cinta oráculo se modifica para contener la cadena de z en los cuadros 1 a $|z|$ y símbolos en blanco en los demás cuadros, la cabeza del oráculo se coloca para revisar el cuadro 1, y el control de estado finito se cambia del estado q_c al estado q_r. Este paso deja el contenido de la cinta principal y la posición de su cabeza sin cambios.

La principal diferencia entre una MTD y una MTO está en este tercer tipo de paso, mediante el cual un programa MTO puede *consultar* el oráculo. Si la MTO escribe una cadena de consulta y entonces entra al estado-oráculo de consulta, la cadena de respuesta $z=g(y)$ será devuelta en un solo paso del cálculo. Por lo tanto, esto corresponde a una subrutina hipotética para calcular la función g . El cálculo de un programa MTO *M* sobre una cadena de entrada x depende de x y de la función de oráculo g asociada.

Sea M_g un programa MTO *generado* obtenido de la combinación de M con el oráculo g . Si M_g se detiene para todas las entradas $x \in \Sigma^*$, también puede ser vista como calcular una función $f_M^g : \Sigma^* \to \Gamma^*$, definida exactamente de la misma forma que para una MTD. Se dice que M_g es un *programa MTO de tiempo polinómico* si existe un polinomio p tal que M_g se detiene con $p(|x|)$ pasos para cada entrada $x \in \Sigma^*$.

Sea R y R' dos relaciones de cadena sobre Σ . Una *reducción de Turing en tiempo polinómico* de R a R' es un programa MTO *M* con un alfabeto de entrada Σ tal que, para cada función $g : \Sigma^* \to \Sigma^*$ que produce R' , el programa generado M_g es un programa MTO de tiempo polinómico y la función f_M^g calculada por M_g produce R . Si existe un reducción de R a R' , escribimos $R \propto_T R'$, que se lee como R *Turing-reduce a* R' . La relación $\propto_T$ como $\propto$, es transitiva.

Una relación de cadena R es NP-difícil si existe algún lenguaje L NP-completo tal que $L \propto_T R$. Un problema de búsqueda Π (bajo el sistema de codificación e) es NP-difícil si la relación de cadena $R[\Pi, e]$ es NP-difícil. Informalmente, esto se interpreta diciendo que un problema de búsqueda Π es NP-difícil si existe algún problema NP-completo Π' que se Turing-reduce a Π . Es fácil ver que si una relación de cadena R (o problema de búsqueda Π) es NP-difícil, entonces no puede ser resuelto en tiempo polinómico a menos que $P = NP$.

Por transitividad de $\propto_T$, si R es cualquier relación de cadena NP-difícil y si R Turing-reduce a la relación de cadena R' , entonces R' también es NP-difícil. Además, por la asociación de lenguajes con relaciones de cadenas y problemas de decisión con problemas de búsqueda, se puede decir que todos los lenguajes NP-completos y todos los problemas NP-completos son NP-difíciles.

Capítulo 3

Algoritmos de aproximación para conjuntos dominantes conectados

En este capítulo se presenta una introducción a los algoritmos de aproximación, dando algunas definiciones básicas que nos permiten conocer cuándo un algoritmo de aproximación es mejor que otro. Posteriormente se da una introducción al problema del conjunto dominante conectado, se exponen algunas de sus aplicaciones y finalmente se describen algunos de los algoritmos de aproximación centralizados más importantes que ofrecen una solución al problema y se da su factor de aproximación en cada uno de ellos.

3.1. Introducción a los algoritmos de aproximación

Los algoritmos de aproximación se desarrollaron como respuesta a la imposibilidad de resolver una gran cantidad de problemas importantes de optimización discreta en tiempo polinómico, debido a que muchos de estos problemas son NP-difíciles. Por lo tanto, a menos que de P sea igual a NP, no existen algoritmos rápidos para encontrar las soluciones óptimas.

Debido a lo anterior, no podemos tener al mismo tiempo algoritmos que (1) encuentren soluciones óptimas (2) en tiempo polinómico (3) para cualquier instancia del problema. Por lo anterior, en cualquier enfoque al menos uno de estos requisitos debe ser relajado para hacer frente al problema de optimización a tratar.

Uno de los enfoques relaja el requisito (3) “para cualquier instancia”, y se enfoca a casos especiales del problema, encontrando para estos casos algoritmos de tiempo polinómico. Este enfoque sólo es útil cuando las instancias a resolver caen dentro de los casos especiales para

los que se creó el algoritmo que trabaja de forma rápida, pero esto no ocurre con frecuencia.

Un segundo enfoque, relaja el requisito (2) “resolver en tiempo polinómico”. Su objetivo es encontrar las soluciones óptimas de los problemas explorando todo el conjunto de soluciones posibles. Se debe tener presente que este enfoque nos será de utilidad únicamente si se tiene el tiempo, digamos varios minutos o incluso horas, para encontrar la solución óptima.

El enfoque más común relaja el requisito (1) “encontrar soluciones óptimas”. En este enfoque nos conformamos con encontrar soluciones suficientemente buenas en un tiempo bastante corto (de orden polinómico). Este enfoque ha producido una gran cantidad de heurísticas y meta-heurísticas tales como el recocido simulado, algoritmos genéticos, búsqueda tabú, etc; que suelen dar buenos resultados.

En especial, los algoritmos de aproximación caen dentro de este último enfoque, relajando el requisito de encontrar soluciones óptimas. Su principal paradigma es el intercambiar optimalidad en favor de tratabilidad [15], pero intentando sacrificar tan poca optimalidad y ganar tanta eficiencia como sea posible.

Los algoritmos de aproximación hacen un estudio matemático para determinar de manera precisa que tan lejos la solución, en el peor de los casos, puede caer. Este estudio implica derivar estimaciones sobre el valor de la solución óptima.

El concepto formal de un algoritmo de aproximación fue formulado por Garey et al. [12] y más tarde por Johnson [18]. El concepto formal se define de la siguiente manera:

Definición 4 *Un algoritmo ρ -aproximado para un problema de optimización es un algoritmo de tiempo polinómico que para todas las instancias del problema obtiene una solución cuyo valor está dentro de un factor ρ del valor de una solución óptima.*

Para un algoritmo ρ-aproximado, ρ es el *factor de aproximación del algoritmo*. Donde, $\rho > 1$ para problemas de minimización, mientras que $\rho < 1$ para problemas de maximización. Así, un algoritmo 1/2-aproximado para un problema de maximización es un algoritmo de tiempo polinómico que siempre devuelve una solución cuyo valor es por lo menos la mitad del valor óptimo.

Entre las razones que podemos encontrar para estudiar algoritmos de aproximación se encuentran: porque se requieren algoritmos rápidos para un problema NP-difícil o más aún por

el soporte matemático cuando se estudia la heurística. Ya que, las heurísticas y meta heurísticas pueden funcionar bien, pero muchas veces no entendemos por qué lo hacen. Sin embargo los algoritmos de aproximación, permiten demostrar a través de su estudio matemático que tan bien la solución, para cualquier instancia dada, se comportará y por qué lo hace; e incluso nos dan una idea de los tipos de instancias en que la heurística no funcionaría bien.

Hasta aquí, podemos resumir que un algoritmo de aproximación es una heurística a la que analizamos matemáticamente, para obtener el valor de la métrica ρ que es la que determina que tan lejos la solución obtenida por el algoritmo, en el peor de los casos, estaría de la solución óptima y lo más interesante; aún sin conocer el verdadero valor de la solución óptima. Es claro que en la práctica, preferimos los algoritmos de aproximación que nos proporcionen el ρ que se encuentre dentro de un pequeño porcentaje del óptimo. También, es frecuente que en la práctica los algoritmos de aproximación den lugar a heurísticas que regresan soluciones mucho más cercanas al óptimo indicado en su factor o garantía ρ de desempeño.

En el caso de algunos problemas, es posible obtener muy buenos algoritmos de aproximación, de hecho estos problemas tienen *esquemas de aproximación de tiempo polinómico*.

Definición 5 *Un esquema de aproximación de tiempo polinómico (PTAS, Polynomial Time Approximation Scheme) es una familia de algoritmos* $\{ A_\varepsilon \}$*, donde hay un algoritmo para cada* $\varepsilon > 0$ *, tal que* $\{ A_\varepsilon \}$ *es un algoritmo* $(1+\varepsilon)$*-aproximado para problemas de minimización o un algoritmo* $(1-\varepsilon)$*-aproximado para problemas de maximización.*

Muchos problemas tienen esquemas de aproximación de tiempo polinómico. Por ejemplo, el problema de la mochila o el problema del vendedor ambulante con métrica euclidiana [29].

3.2. Problema del Conjunto Dominante Conectado

El origen del concepto *conjunto dominante* puede remontarse a la década de 1850 [30], en esa época se consideró el siguiente problema entre los amantes de ajedrez en Europa: determinar el mínimo número de reinas que se pueden colocar en un tablero de ajedrez de tal forma que todos los cuadros sean atacados u ocupados por una reina. Se encontró que cinco es el número

mínimo de reinas que pueden dominar todas las casillas de un tablero de ajedrez de 8 por 8. El problema de las cinco reinas se formula como un conjunto dominante de una gráfica $G=(V,E)$, donde los vértices corresponden a las casillas del tablero de ajedrez, y $(u,v)\in E$ si y solo si la reina se puede mover de la casilla correspondiente u a la casilla correspondiente v.

Un *conjunto dominante*, es un subconjunto $V'\subseteq V$ tal que, para todo $u\in V-V'$ existe al menos un nodo $v\in V'$ tal que $(u,v)\in E$.

Un *conjunto dominante conectado* (*CDC*), es un conjunto dominante de *G* que forma un subgráfica conexa.

Un *conjunto dominante conectado mínimo* (CDCM), es el CDC de cardinalidad mínima. Encontrar un CDCM es NP-difícil [13].

El problema del conjunto dominante conectado mínimo se estudió en [13], donde se mostró una reducción del problema del conjunto cubridor al problema CDCM. Este resultado implica que para cualquier $0<\varepsilon<1$, no existen algoritmos de tiempo polinómico con factor de desempeño menor o igual a $(1-\varepsilon)H(\Delta)$ a menos de que $NP\subset DTIME[n^{O(\log\log n)}]$ [22], donde H es la función armónica $H(n)=1+1/2+...+1/n\in O(\log n)$. Cuando se restringe a gráficas de discos unitarios (UDG), el problema CDCM sigue siendo NP-difícil [7]. Por lo tanto calcular un CDCM en una gráfica de esferas unitarias también es NP-difícil.

En [1, 4, 28] se demostró que el problema de encontrar un CDCM en UDGs tiene un factor de desempeño constante. También mediante estrategias distribuidas, Butenko y Ursulenko [3] y Kim et al. [19] dieron aproximaciones constantes para CDCM en UBGs, en [19] también se tiene un factor de desempeño constante para CDCM en UBGs pero con una estrategia centralizada.

En cuanto a algoritmos centralizados para CDCs en UDGs Cheng et al. [5] dio un esquema de aproximación de tiempo polinómico (PTAS), esto es, para cualquier $\varepsilon>0$, existe una ($1+\varepsilon$)-aproximación de tiempo polinomial. Este método no puede ser generalizado para obtener PTAS para CDCM en UBGs [31], sin embargo Zhang et al. [31] presentó un PTAS para UBGs donde utiliza un nuevo método para analizar el factor de desempeño, de hecho el método puede ser utilizado para calcular CDCs para cualquier gráfica de esferas n-

dimensional. El primer paso de este PTAS requiere un algoritmo ρ -aproximado para el problema del CDCM en UBGs. El número de operaciones está determinado por el parámetro de aproximación ε y ρ. Por lo tanto, aunque existe una PTAS para este problema, el diseño de algoritmos de aproximación con buenos factores de aproximación es aún muy importante.

Es importante aclarar que las heurísticas propuestas para gráficas de discos unitarios funcionan bien para las gráficas en general, pero en este caso su respectivo análisis de rendimiento no es válido.

3.3. Aplicaciones de los conjuntos dominantes conectados

En la última década el problema del mínimo conjunto dominante conectado ha sido de gran relevancia en el estudio de las redes inalámbricas [4, 9, 28 30]. Existen diversas aplicaciones en este enfoque, entre ellas encontramos la transmisión de información a todo el conjunto de nodos inalámbricos que forman parte la red, en donde un nodo emisor envía información a través de un mensaje a sus nodos vecinos y los nodos vecinos que pertenecen al conjunto dominante conectado se encargan de retransmitir el mensaje. Para cada nodo que recibe el mensaje: (1) si pertenece al conjunto dominante entonces también es responsable de retransmitir el mensaje, mientras que (2) todos los nodos que no pertenecen al conjunto dominante conectado no son responsables de retransmitir el mensaje [9]. La presente tesis trata con este problema y lo detalla en el siguiente capítulo.

Los algoritmos que construyen un CDC en una red ad hoc se pueden clasificar principalmente en dos categorías: *algoritmos centralizados* que dependen de la información de toda la red y *algoritmos descentralizados* o *distribuidos* que dependen únicamente de la información local. Los algoritmos centralizados usualmente forman CDC más pequeños que los algoritmos distribuidos, pero sus aplicaciones son limitadas debido al alto costo de su mantenimiento.

En las siguiente secciones de este capítulo se analiza el funcionamiento de algunos algoritmos de aproximación centralizados para la construcción de CDCs y en el capítulo 4 se analizan las estrategias distribuidas.

3.4. Algoritmo de Guha y Khuller

Guha y Khuller [14] propusieron dos estrategias para la construcción de CDC en gráficas generales. En su trabajo presentan dos algoritmos voraces, cada uno con su respectivo factor de aproximación. Esto algoritmos han servido como base en la implementación de muchos otros algoritmos. El primero forma un CDC haciendo crecer un árbol a partir de uno de los nodos, y nos muestran que su factor de aproximación es $2(1 + H(\Delta)) |opt|$. El segundo algoritmo es una mejora del primero, este algoritmo se lleva a cabo en dos etapas. En la primera etapa se construye un conjunto dominante haciendo crecer componentes por separado y en la segunda etapa se conecta a los nodos del conjunto dominante, su factor de desempeño es $(3 + ln (\Delta)) |opt|$. Das y Bharghavan [9] implementaron estos algoritmos de manera distribuida.

El funcionamiento básico del primer algoritmo es el siguiente, al inicio se marcan todos los nodos en blanco y se selecciona el vértice de mayor grado. El vértice seleccionado se marca en negro y todos sus vecinos se marcan en gris. Después, de forma iterativa se selecciona un nodo gris o un par de nodos (un nodo marcado en gris y uno de sus vecinos marcado en blanco), cualquiera de las dos opciones que cumpla con el requisito de tener el mayor número de nodos vecinos marcados en blanco. El nodo seleccionado o par de nodos seleccionados se marcan en negro y sus vecinos blancos en gris. El algoritmo termina después de que todos los vértices han sido marcados en gris o negro. Finalmente, el conjunto de nodos negros son los que forman parte del conjunto dominante conectado. El algoritmo forma un CDC de tamaño a lo más $2(1 + H(\Delta)) |opt|$, donde H es una función armónica y opt se refiere a una solución óptima, es decir un conjunto dominante conectado mínimo.

En el segundo algoritmo se define una pieza como un componente negro conectado o un bien un nodo blanco. El algoritmo también comienza marcando todos los vértices en blanco y se lleva a cabo en dos etapas. En la primera etapa se selecciona de forma iterativa el nodo blanco que cause la mayor reducción en el número de piezas. Una vez que se ha seleccionado un nodo, el nodo es marcado de negro y sus vecinos blancos se marcan en gris. La primera etapa termina cuando ya no quedan nodos blancos. Después de la primera etapa existen a lo más $|opt|$ número de componentes negros conectados. En la segunda etapa se construye un árbol de Steiner que conecta todos los nodos negros coloreando cadenas de dos nodos grises en negro. El tamaño del CDC formado por todos los nodos negros es a lo más $(3 + ln(\Delta)) |opt|$.

3.5. Algoritmo de Ruan

El algoritmo de Ruan et al. [26] es un algoritmo de una sola etapa con factor de desempeño $(2 + ln(\Delta))\ |opt|$. El algoritmo es una modificación al segundo algoritmo presentado por Guha y Khuller [14].

Al igual que el algoritmo en [14] comienza marcando en blanco todos los nodos de la gráfica. Luego, de forma iterativa elige un nodo gris o blanco de tal forma que al marcar en negro el nodo elegido y sus vecinos blancos en gris, se reduzca al máximo el valor de la función potencial.

La función potencial se define de la siguiente manera, $f(C) = p(C) + q(C)$, donde C es el subconjunto de vértices negros, $p(C)$ es el número de componentes conectados de la subgráfica formada por los vértices en negros y q(C) es el número de componentes de la subgráfica $(V, D(C))$.

Al inicio $f(C) = |V|$, ya que $C = \emptyset$. En el primer paso se elige el nodo de mayor grado. En los siguientes pasos se selecciona un nodo x de tal forma que $f(C)$ - $f(C \cup \{x\})$ sea maximizado. Entonces se marca el nodo x en negro y en gris sus vecinos blancos. Finalmente el algoritmo termina cuando $f(C) = 2$. Los nodos en el conjunto C son los nodos que forman parte del conjunto dominante conectado.

3.6. Algoritmo de Min

Min et al. [23] presentó un algoritmo de aproximación para el problema de CDCM en UDGs, al cual tiene un factor de aproximación de $6.8\ |opt|$. El algoritmo se basa en la idea popular de construir un CDC en dos etapas, en la primera etapa construye un conjunto dominante y en la segunda etapa conecta los nodos del conjunto dominante formando así el CDC. La idea principal en este trabajo es utilizar un árbol de Steiner para hacer el trabajo de la segunda etapa, mientras que para obtener el conjunto dominante en la primera etapa se construye un conjunto independiente maximal.

Su factor de aproximación se calcula en base al algoritmo presentado para calcular un árbol de Steiner con un mínimo número de nodos Steiner (ST-MSN, Steiner Tree with Minimum

Number of Steiner Nodes) el cual tiene una 3-aproximación, por otro lado asume que el conjunto independiente maximal calculado en la primera etapa satisface el Lema 1.

Lema 1 *En cualquier gráfica de discos unitarios, el tamaño de cada conjunto independiente maximal tiene una cota superior de 3.8 |opt| + 1.2 donde opt es el tamaño del conjunto dominante conectado de tamaño mínimo en este UDG.*

Por lo tanto, ya que se utilizan a lo más *3 |opt|* nodos de Steiner en la segunda etapa y por el Lema 1, el conjunto dominante conexo resultante esta acotado en su tamaño por *6.8 |opt|*. A continuación se describe el algoritmo ST-MSN.

La entrada del algoritmo es un conjunto independiente maximal cuyo nodos están marcados en negro y los demás nodos de la gráfica en gris. Un componente negro es un componente conexo de la subgráfica producido por los nodos negros.

1. De manera iterativa, mientras exista un nodo gris u adyacente a por lo menos tres componentes negros.
    - Marcar el nodo u en negro.
2. De manera iterativa, mientras exista un nodo gris u adyacente a por lo menos dos componentes negros.
    - Marcar el nodo u en negro.
3. Finalmente, el conjunto de nodos negros forman un CDC

3.7. Algoritmo C-CDC-UBG de Kim

Kim et al. [19] desarrollaron un algoritmo de aproximación centralizado denominado C-CDC-UBG. El algoritmo C-CDC-UBG construye un conjunto dominante conectado (CDC) en gráficas de esferas unitarias (UBG) y tiene un factor de aproximación de *14.937 |opt| + 1.083*. El algoritmo se ejecuta en dos etapas, en la primera etapa calcula un conjunto independiente maximal o MIS (por sus siglas en ingles) M_0 de tal manera que para cualquier $M \subset M_0$, la distancia entre *M* y $M_0 \setminus M$ es de exactamente dos saltos. En la segunda etapa conecta los nodos del MIS en M_0 seleccionando algunos nodos en $V \setminus M_0$ usando una estrategia voraz simple.

La estrategia en C-CDC-UBG para construir el conjunto dominante conectado *C* es crecer iterativamente *C* añadiendo en cada paso un nodo no-MIS *v* junto con sus nodos MIS adyacentes, el requisito para seleccionar a *v* es que | $M_{v,C}$ | sea máximo. Recordemos que *M_{v,C}* es el conjunto de nodos MIS adyacentes a *v* que aún no pertenecen al conjunto dominante conectado *C*.

El algoritmo C-CDC-UBG comienza la primera etapa con $M_0 = \{r\}$, donde *r* es el nodo con el mayor grado de la gráfica, y el conjunto *W = N(r)*, *W* durante el algoritmo es el conjunto de nodos adyacentes a M_0 . Denominemos como nodos *rastreados* a los nodos en los conjuntos M_0 y W. Después, de manera iterativa elige un nodo $x \in N(W)$ de tal manera que |N(x) ∩ *el conjunto de nodos no rastreados*| sea maximizado. Añadimos *x* al conjunto M_0 y el conjunto de nodos no rastreados en *N(x)* a *W*, observe que una vez añadidos a *W*, los nodos son marcados como rastreados al igual que *x*. La primera etapa del algoritmo termina cuando todos los nodos han sido marcados como rastreados.

La segunda etapa del algoritmo comienza con el conjunto $C = \{r\}$, llamemos a los nodos en *C* como nodos *revisados* y a *M* como el conjunto de nodos en MIS que aún no se han marcado como revisados o bien que no están en *C*. Entonces, de manera iterativa selecciona un nodo $v \in N(C)$ de tal manera que $M_{v,C} = \{máx \mid M_{x,C} \mid \mid x \in N(C)\}$. Una vez que se ha elegido *v* es añadido al conjunto *C* al igual que $M_{v,C}$ y $M = M \setminus M_{v,C}$. La segunda etapa del algoritmo termina cuando todos los nodos en el MIS han sido marcados como revisados, es decir $M = \emptyset$. Finalmente, los nodos en *C* forman el conjunto de nodos dominantes conectado.

Capítulo 4

Aproximaciones distribuidas

En este capítulo se presenta una introducción a las redes móviles ad hoc (MANETs) y se presenta un modelo matemático que permite el estudio de estas redes. También se habla de algunos de los problemas bajo este contexto, enfatizando el problema de la diseminación de información. El problema de la diseminación de información se plantea como un problema de optimización llamado el problema del árbol de inundación, que se describe en la sección 4.1.2, y que es equivalente al problema del conjunto dominante conectado. Por lo tanto, encontrar soluciones para el problema del conjunto dominante conectado equivale a resolver el problema de la diseminación de información. Ya que tanto el problema del árbol de inundación y el problema del conjunto dominante conectado son NP-difíciles, entonces se hace uso de algoritmos de aproximación para resolver el problema. Finalmente se describen algunos de los algoritmos de aproximación distribuidos que construyen conjuntos dominantes conectados.

Redes móviles ad hoc

Las *redes inalámbricas* constan de un conjunto de nodos que pueden comunicarse entre ellos mediante canales inalámbricos. Algunas redes de este tipo están formadas por una red dorsal alambrada y sólo el último salto hacia un nodo de la red se realiza mediante un canal inalámbrico. En otro tipo de redes todos los canales son inalámbricos, ejemplo de este tipo de redes son las *redes ad hoc*.

Las redes ad hoc son sistemas autónomos conformadas por un conjunto de nodos móviles que hacen uso de transmisiones inalámbricas para comunicarse entre sí. Los nodos en las redes ad hoc forman una red sin una infraestructura física preestablecida, se auto-organizan, *auto-configuran,* y auto-controlan. Por este motivo, establecer una red de este tipo en cualquier

lugar y en cualquier momento resulta sencillo y hasta económico. Las redes ad hoc son utilizadas por diferentes comunidades de usuarios, tales como investigadores, militares, empresarios, estudiantes y también en servicios de emergencia.

Debido a su administración descentralizada, en las *redes móviles ad hoc* (*MANETs*) es mucho más conveniente hacer uso de a*lgoritmos distribuidos*, ya que el tamaño de la red puede ser un problema si se intenta hacer uso de algoritmos centralizados. Por ejemplo si hacemos uso de alguno de los algoritmos vistos en el Capítulo 3 que construyen una red dorsal virtual, nos enfrentaríamos con el problema de que al menos un nodo debe conocer la información de la topología completa de la gráfica, entonces el número de mensajes que se requiere intercambiar para lograr este objetivo puede crecer rápidamente dependiendo del tamaño de la red y peor aún si los nodos son altamente móviles nos enfrentamos con el problema de que la información que reciba el nodo acerca de los demás nodos pueda estar ya obsoleta.

Cada nodo en una red ad hoc actúa como un ruteador, para lograr el intercambio de información. Si un nodo fuente no puede enviar directamente un paquete a un nodo destino debido a su rango de transmisión limitado, entonces el nodo fuente envía el paquete a nodos intermedios y los nodos intermedios reenvían el paquete hacia el nodo destino.

Algunos de los algoritmos de ruteo más importantes para redes móviles ad hoc, tal como AODV [24] y DSR [17] dependen de la técnica de *inundación* (*flooding*) para *diseminar información* a través de la cual se obtiene su información de ruteo. En esta técnica cada que un nodo recibe por primera vez un paquete lo retransmite a todos sus nodos vecinos. Ya que es muy probable que las señales se traslapen con otras en un área geográfica, llevar a cabo la retransmisión de paquetes usando la técnica *flooding* resulta ser altamente costoso, ya que da lugar a una *alta redundancia* de la información en cada nodo al recibir un paquete repetidas veces por parte de sus vecinos, *alta contención* ya que probablemente nodos retransmitiendo están muy cercanos unos de otros y mayor probabilidad de *colisiones*. Los problemas anteriores de la técnica *flooding* son ahora conocidos como el problema de *tormenta de transmisión* (*the broadcast storm problem*) estudiado en [27].

En la siguiente sección se presenta el modelo matemático de las redes ad hoc que permite hacer el análisis para obtener el factor desempeño de los algoritmos de aproximación en este enfoque.

4.1.1. Modelo matemático de las redes ad hoc

Una red ad hoc se puede modelar con una gráfica dinámica $G(V^t, E^t)$, donde cada terminal móvil es representada por un vértice del conjunto V^t y para cada par de nodos u, $v \in V^t$, $(u, v) \in E^t$ si y solo si los nodos u y *v* se encuentra dentro de su rango de transmisión uno del otro. Los conjuntos E^t y V^t varían con el tiempo debido a que los nodos en las redes ad hoc son móviles y por tanto su topología es dinámica, además de que en cualquier momento un nodo puede dejar de formar parte de la red, debido a que su batería se terminó por ejemplo.

Hasta hace algunos años era común modelar el rango de comunicación de un nodo en una red inalámbrica como un disco centrado en el nodo con radio igual al radio de su rango de transmisión. Cuando el rango es el mismo para todos los nodos la red tiene las propiedades de una *gráfica de discos unitarios* (*UDG, Unit Disk Graphs*), donde existe una arista entre dos nodos si y solo si la distancia entre ambos nodos es menor o igual a una unidad. Muchos de los algoritmos CDC usan las propiedades de las UDGs para probar sus respectivas garantías de desempeño.

Más recientemente se está utilizando otro tipo de gráficas para modelar las redes ad hoc en espacios de tres dimensiones [2, 3, 19, 31, 32], estas gráficas son llamadas *gráficas de esferas unitarias* (*UBG, Unit Ball Graphs*), y llevan el estudio de las redes ad hoc un paso más adelante hacia la realidad.

Debido a que las UDGs son instancias especiales en las cuales la altura de los nodos es la misma, todos los problemas que son NP-difíciles en UDGs también los son en UBGs. Por supuesto, el problema de encontrar un conjunto dominante conectado mínimo es también NP-difícil en UBGs.

4.1.2 Árbol de inundación óptimo en redes ad hoc

Cuando se requiere diseminar información a todos los nodos de la red ad hoc, el método más sencillo es hacer uso de la técnica de inundación en la que cada nodo retransmite un paquete la primera vez que le llega, pero como se vio en la sección 4.1 resulta ser altamente costoso

[27]. La solución a este problema es optimizar el número de retransmisiones que se hacen para entregar un paquete a todos los nodos de la red, para lograr este objetivo se busca construir un árbol tal que que el número de nodos hojas sea máximo. Ya que los nodos del árbol responsables de retransmitir los paquetes a diseminar por toda la red serían entonces los nodos internos. Por lo tanto, el número de retransmisiones para diseminar un paquete es igual al número de nodos internos del árbol. Encontrar este árbol es planteado como el problema del *árbol de inundación óptimo* [21],

Definición 6 *Un árbol de inundación es un árbol que cubre todos los nodos en una gráfica*.

Definición 7 *Un árbol de inundación óptimo es un árbol de inundación con el costo mínimo. El costo de un árbol de inundación en una red ad hoc se define como el número de retransmisiones para entregar un paquete a todos los nodos.*

El problema del árbol de inundación óptimo es el mismo que el problema del árbol de expansión (*Spanning Tree*) con el máximo número de hojas en una red cableada, el cual maximiza el número de nodos hojas reduciendo de esta manera los enlaces requeridos para inundar un paquete. Dada la información de la topología completa de la red, un nodo podría encontrar fácilmente un árbol de expansión, sin embargo, en una red ad hoc no es una tarea sencilla.

Lim y Kim [21] descubrieron que el problema del árbol de inundación óptimo es similar al problema del conjunto dominante conexo mínimo (CDCM) y lo demostraron probando el Teorema 5.

Teorema 5 *Dado una gráfica G(V, E), sea T un árbol de inundación óptimo de G y M un CDCM. Entonces* $|T| = |M|$ *o* $|T| = |M| + 1$ *donde* $|T|$ *es el número de nodos internos en el árbol de inundación T y* $|M|$ *es el tamaño de M.*

También, probaron el Teorema 6, mostrando así que el problema del árbol de inundación óptimo es NP-completo.

Teorema 6 *El problema del árbol de inundación óptimo es NP-completo*

Ya que es difícil encontrar un árbol de inundación óptimo, debemos intentar ofrecer soluciones a través de uno de los tres diferentes enfoques vistos en la sección 3.1. A lo largo de las siguientes secciones se analiza el funcionamiento de algunos de los *algoritmos de aproximación*

distribuidos que construyen CDCs, o una *red dorsal virtual* (*virtual backbone*).

4.2. Algoritmo de Wan

En el trabajo de Wan et al. [28] se presenta un algoritmo de aproximación distribuido para calcular un CDC en UDGs, su factor de aproximación es de a lo más $8|opt| + 1$, su complejidad en el tiempo es $O(n)$, y su complejidad de mensajes es $O(n \log n)$, donde n es el número de nodos y *opt* el tamaño de un CDCM. El algoritmo se ejecuta en tres etapas descritas a continuación.

- Etapa de *selección de un líder*, en esta etapa el algoritmo elige un nodo v como líder y construye un árbol de expansión con raíz en v. Para elegir a v se puede utilizar cualquier estrategia de elección de un líder como la mostrada en [6].
- Etapa de *cálculo de nivel*, en esta etapa cada nodo identifica el nivel en el que se encuentra dentro del árbol. El nodo v comienza anunciando su nivel igual a 0, luego cada nodo que recibe un mensaje por parte de su nodo padre dentro del árbol, calcula su nivel incrementando en 1 el nivel anunciado por su padre y envía un mensaje anunciando su propio nivel. Al final de esta etapa cada nodo conoce los niveles e identificadores de sus vecinos.
- Etapa de *coloreado*, al comenzar esta etapa todos los nodos son marcados en blanco, luego el nodo raíz v se marca en negro y transmite un mensaje DOMINADOR a sus vecinos, entonces cada nodo actúa de acuerdo a los siguientes criterios:
    - Si un nodo blanco recibe un mensaje DOMINADOR por primera vez, se marca en gris y transmite un mensaje DOMINADO.
    - Cuando un nodo blanco recibe un mensaje DOMINADO de todos sus vecinos de nivel inferior, se marca en negro y transmite un mensaje DOMINADOR.
    - Cuando un nodo gris recibe un mensaje DOMINADOR por primera vez de alguno de sus nodos hijos dentro del árbol, que nunca ha enviado un mensaje DOMINADO, se marca en negro y transmite un mensaje DOMINADOR.
    - Si un nodo negro tiene más alto nivel que sus vecinos y todos sus vecinos son

negros, entonces se marca ahora en gris y transmite el mensaje DOMINADOR.

Al final de esta etapa, los nodos marcados en negro son los nodos que forman el CDC. Ya que todos los nodos han sido marcados en gris o negro y cada nodo gris es adyacente a al menos un nodo negro existe un conjunto dominante, y ya que entre cualesquiera dos nodos negros existe un camino de nodos negros entonces los nodos negros forman un CDC.

4.3. Algoritmo de Li

Li et al. [20] presentaron un algoritmo de aproximación llamado S-MIS, el algoritmo construye CDCs en UDGs mediante la construcción de un conjunto independiente maximal en una primera etapa y de un árbol de Steiner con un mínimo número de nodos de Steiner (*ST-MSN, Steiner tree with minimum number of Steiner nodes*) en una segunda etapa.

El factor de aproximación de este algoritmo es de $(4.8 + ln\ 5)\ |opt| + 1.2$, el cual no se ha podido mejorar hasta la fecha. El calculo del valor del factor de aproximación se basa en el Lema 1 de la sección 3.6 para el tamaño del MIS que es $3.8\ |opt| + 1.2$ y en el calculo del valor del factor de aproximación para el ST-MSN que es igual a $(1 + ln\ 5)\ |opt|$. Por lo tanto, el algoritmo S-MIS obtiene CDCs de tamaño acotado por $(4.8 + ln\ 5)\ |opt| + 1.2$, donde $|opt|$ es el tamaño del CDCM.

Para construir el MIS de la primera etapa, se pueden utilizar alguna de las estrategias mostradas en [4] o en [28]. La segunda etapa construye un CDC basado en el MIS, el cual tiene todos sus nodos marcados en negro y los demás nodos de la gráfica están marcados en gris. El funcionamiento del algoritmo para la segunda etapa es el siguiente:

Un *componente negro-azul* es un componente conexo de la subgráfica producida únicamente por los nodos negros y azules. Cada nodo azul o negro tiene un z-valor que indica el componente negro al que pertenecen y cada nodo gris tiene dos valores, un y-valor que indica el número de nodos negros adyacentes en diferentes componentes negro-azul y un segundo valor que es su identificador. El nodo con el y-valor más alto se toma como el de *mayor peso*. En caso de empate se toma el identificador más pequeño.

Un nodo gris u es adyacente a un componente negro-azul si es adyacente a un nodo negro del componente y se dice que *u compite* con otro nodo gris *v* si *u* y *v* son adyacentes al mismo

componente negro-azul.

Un nodo gris u cambia su color a azul si y solo si u es clasificado como el de mayor peso de todos sus competidores. Cada nodo gris conserva dos listas, la de sus competidores y una lista negra. La lista negra contiene los nodos negros adyacentes con sus z-valores, la lista de sus competidores contiene todos sus competidores y su lista negra.

1. Cuando un nodo u se marca en azul, todos sus componentes adyacentes se unen en un único componente y por tanto sus z-valores se igualan al enviar u un mensaje UPDATE(u) a todos sus vecinos.
2. Cuando un nodo v recibe un mensaje UPDATE(u), actualiza su z-valor, envía un mensaje COMPLETE(u) y pasa el UPDATE(u) a sus vecinos.
3. Cuando un nodo gris recibe un mensaje UPDATE(u) actualiza su lista negra y la lista de sus competidores y envía un mensaje COMPLETE(u) a sus vecinos.

4.4. Algoritmo de Butenko y Ursulenko

Butenko y Ursulenko [3] propusieron un algoritmo de aproximación con un factor constante igual a 22, para calcular CDCs en UBGs. El algoritmo construye un MIS I, y durante el proceso conecta los nodos que se van añadiendo a I con algunos vértices que se marcan como padres. Al final del algoritmo se obtiene el CDC D que contiene todos los vértices en I y los vértices marcados como padres. Este algoritmo consiste de tres etapas descritas a continuación.

- Etapa de *inicialización*, en esta etapa se selecciona un vértice r con el mayor grado de la gráfica y se añade r a los conjuntos I y D. Los nodos en N(r) y r se marcan como *explorados*.
- Etapa de *construcción*, en esta etapa se añade a los conjuntos I y D un nodo x no explorado de manera iterativa. El nodo x que se elige debe cumplir con ser adyacente a por lo menos un nodo en el vecindario de los nodos explorados y debe tener el mayor número de vecinos no explorados. Para conservar la conectividad se añade también un nodo y en D. El nodo y elegido debe ser un vecino explorado de x, y de ser posible que sea un nodo ya marcado como padre por algún otro nodo. Luego x marca a y como su vértice padre. Esta etapa finaliza cuando ya no quedan nodos no explorados.

- Etapa de *mejora*, en esta última etapa se intenta reducir el tamaño del conjunto dominante conectado. Esta etapa se ejecuta únicamente cuando todos los nodos padres añadidos a D en la etapa de construcción, son padres de un único nodo en I. En este caso se añade a D un nodo u elegido del vecindario de I tal que u tenga el mayor número de vecinos en I y se eliminan de D todos los nodos padres de los nodos en I adyacentes a u.

4.5. Algoritmo D-CDC-UBG de Kim

En el trabajo presentado por Kim et al. [19] se desarrollo un algoritmo de aproximación distribuido para calcular conjuntos dominantes conectados en gráficas de esferas unitarias, el algoritmo fue nombrado D-CDC-UBG y se originó a partir del algoritmo visto en la sección 3.7 (es decir, misma estrategia voraz) presentado en el mismo trabajo y por tanto conserva el mismo factor de aproximación que es $14.937\,|opt| + 1.083$ [19].

D-CDC-UBG es un algoritmo de dos etapas, en la primera etapa obtiene un MIS de manera distribuida mediante la ejecución del algoritmo D-MIS-UBG que es un algoritmo de coloreado simple y en la segunda etapa conecta los nodos del MIS añadiendo algunos nodos y formando de esta manera un CDC.

El algoritmo D-MIS-UBG comienza marcando todos los nodos en blanco luego de manera iterativa marca un nodo blanco en negro y sus nodos vecinos en gris. Cuando el algoritmo finaliza, el conjunto de nodos negros forman parte del MIS. El funcionamiento del algoritmo es el siguiente:

1. Al inicio, para cada nodo de la red el algoritmo establece su conjunto de nodos hijos igual al conjunto de sus vecinos. Luego, selecciona un nodo r con el máximo grado, colorea r de negro y sus nodos hijos de gris. Para cada uno de sus nodos hijos x, establece $P(x) = r$ y elimina de $C(x)$ a todos los nodos en $N[r]$.

2. r transmite un mensaje ASK a todos los nodos blancos adyacentes a un nodos gris. Cuando un nodo que no es blanco recibe el mensaje lo retransmite a sus nodos vecinos. Cuando un nodo blanco x recibe el mensaje envía de regreso un mensaje REP(x, $m(x)$) a r, donde $m(x)$ es el número de nodos blanco en $N[x]$.

3. Cuando un nodo x que no es blanco recibe mensajes REP de todos sus nodos hijos, selecciona uno de sus nodos hijos y tal que $m(y)$ sea máximo, entonces guarda la información de $m(y)$ en $m(x)$ y el identificador del nodo y en $Index(x)$, y envía un mensaje REP(x, $m(x)$) a su nodo padre.
4. Cuando el nodo raíz r recibe mensajes REP de todos sus nodos hijos, selecciona un nodo x de sus hijos tal que $m(x)$ sea máximo. Si $m(x) = 0$, entonces el conjunto de nodos negros es un MIS y r finaliza D-MIS-UBG. De lo contrario, r envía un mensaje JOIN al nodo x.
5. Cuando un nodo x recibe un mensaje JOIN, si x es gris, entonces retransmite el mensaje al nodo en $Index(x)$. Si x es blanco, entonces x se colorea de negro y todos sus nodos hijos de color blancos se colorean de gris. x establece al nodo emisor del mensaje JOIN como su padre, y cada nuevo nodo coloreado de gris establece a x como sus padre. Por último, todos los nodos n recién coloreados envían un mensaje DELETE a sus vecinos y en caso de aplicar, los nodos que reciben el mensaje DELETE eliminan al nodo n del conjunto de sus hijos. El algoritmo se repite desde el paso 2.

La segunda etapa tiene por objetivo colorear de azul los nodos que formarán parte del conjunto dominante conectado. El CDC formado por el conjunto de nodos azules se basa en el MIS obtenido por el algoritmo de la primera etapa. El funcionamiento de la segunda etapa se describe a continuación.

1. Tras la ejecución del algoritmo D-MIS-UBG, r transmite un mensaje IGNORE a todos los nodos de la red. Cuando un nodo x recibe un mensaje IGNORE establece el conjunto de nodos grises adyacentes a él como sus nodos hijos.
2. El nodo r se colorea en azul y todos sus nodo hijos establecen al nodo r como su padre y eliminan del conjunto de sus nodos hijos a $N[r]$.
3. r transmite un mensaje ASK a todos los nodos grises adyacentes a un nodo azul. Cuando un nodo gris y recibe el mensaje envía de regreso un mensaje REP(y, $b(y)$), donde $b(x)$ es el número de nodos negros adyacentes a y.
4. Cuando un nodo azul x recibe todos los mensaje REP de sus hijos, selecciona uno de sus nodos hijos y tal que $b(y)$ sea máximo, almacena el valor de $b(y)$ en $b(x)$ y guarda el

identificador del nodo y en *Index(x)*. Entonces envía un mensaje REP(*x, b(x)*).

5. Cuando el nodo raíz *r* recibe todos los mensajes REP de sus nodos hijos, selecciona uno de sus hijos *x* tal que $b(x)$ sea máximo. Si $b(x) = 0$, entonces el conjunto de nodos azules forma un CDC y el algoritmo finaliza, de lo contrario *r* envía un mensaje JOIN al nodo *x*.
6. Cuando un nodo azul *x* recibe un mensaje JOIN, lo retransmite al nodo en *Index*(x). Si el nodo *x* que recibe el mensaje JOIN es gris, entonces *x* se colorea de azul y el conjunto de todos sus nodos vecinos negros también se colorea de azul. El nodo emisor del mensaje JOIN que *x* recibió se establece como su padre y el conjunto de los vecinos de x recién coloreados de azul establecen al nodo *x* como su padre. Por último, todos los nodos *n* recién coloreados de azul envían un mensaje DELETE a todos sus nodos vecinos para que borren al nodo *n* de sus hijos, si aplica. El algoritmo se repite desde el paso 3.

4.6. Algoritmo PSCASTS de Das

En el trabajo realizado por Das et al. [8] se presentó un esquema de aproximación llamado PSCASTS (*Pseudo-dominating Set Construction And Steiner Tree Spanning*) que mejora el tamaño del CDC formado por los algoritmos basados en la construcción de un MIS en una primera etapa. Su factor de desempeño es de $(4.8 + \ln(5))|opt| + 1.2$ igual al mejor actual reportado utilizando gráficas de discos unitarios (UDG) y la mejor complejidad del tiempo que es $O(D)$, donde D es el diámetro de la red. PSCASTS es un algoritmo distribuido iniciado por múltiples líderes, hace uso del grado de la gráfica e identifica CDCs no triviales y de tamaño más pequeño, aún cuando la distribución de los nodos sensores en la red sea uniforme o casi uniforme.

Su características principales son:

- La construcción de un conjunto seudo-dominante que logra un MIS de cardinalidad más pequeña, ya que se basa en la idea de que cualquiera par de dos nodos más cercanos dentro del MIS, pueden estar a lo más tres saltos de distancia, a diferencia de los anteriores algoritmos que construyen un MIS donde dos nodos más cercanos están a exactamente dos saltos de distancia. Estos MIS con cardinalidad más baja pueden

reducir efectivamente el tamaño de los CDCs.

- Una mejora en la construcción del árbol de Steiner para conectar los nodos en el conjunto seudo-dominante, ya que permite remover nodos en el MIS formado si estos llegan a ser redundantes, es decir si al eliminar el nodo MIS el conjunto dominante sigue siendo conexo. PSCASTS se lleva a cabo en dos etapas

Etapa 1. Construcción de un conjunto seudo-dominante

En esta etapa se construye un conjunto seudo-dominante usando un algoritmo que se basa en el grado y que tiene una complejidad lineal en el número de mensajes. Sea una gráfica $G = (V, E)$.

1. Al inicio del algoritmo todos los nodos se marcan en blanco y cada nodo conoce uno-vecindario y su dos-vecindario.
2. De manera iterativa se selecciona un nodo u que no este marcado en negro, mientras el grado máximo en G sea mayor a 0. El nodo u debe cumplir con tener grado máximo de todos los nodos en su uno-vecindario y dos-vecindario, si u existe entonces:
    - El nodo blanco u se marca en negro.
    - Se eliminan todos los nodos adyacentes a u y las aristas incidentes a ellos de G.
    - Se actualiza el grado de los nodos restantes en G. En caso de empate al elegir a u, se selecciona el nodo que al inicio tenia el mayor grado o bien el que tenga el identificador menor.
3. Todos los nodos en G que no fueron eliminados, excepto los nodos marcados en negro, se marcan en gris.
4. Todos los nodos marcado en negro y los marcados en gris se añaden al conjunto seudo-dominante.

Etapa 2. Mejora en la construcción de un árbol de Steiner

En esta etapa se construye un árbol de Steiner. Un árbol de Steiner para un subconjunto de vértices I de una gráfica, es un árbol que conecta todos los nodos en I usando un conjunto de nodos Steiner formando así un árbol que abarca todos los nodos del conjunto seudo-dominante. Al final de la etapa 2 los nodos marcados en negro, gris y azul forman el conjunto

dominante conexo.

1. Al inicio, todos los nodos marcados en negro o en gris forman un componente separado.
2. De manera iterativa se selecciona un nodo u fuera del conjunto seudo-dominante, mientras todos los nodos marcados en negro y en gris se encuentren en componentes distintos. El nodo u elegido debe cumplir con ser adyacente al mayor número de componentes separados. Si existe un empate, se selecciona el nodo de mayor grado o el que tenga el identificador menor, entonces:
    - Se marca u en azul.
    - Se unen los componentes que conecta u en un solo componente.
3. Los componentes conectados forman el CDC.
4. Para cada nodo v marcado en gris, si v es adyacente a solo un conector del CDC o a dos conectores adyacentes entre si entonces:
    - El nodo v se marca de nuevo en blanco.
    - Se elimina v del CDC.

Capítulo 5

Implementación de los algoritmos

Este capítulo comienza con una breve introducción en la sección 5.1 del entorno de simulación NS2 utilizado para implementar los algoritmos. En la sección 5.2 se habla acerca de los modelos de propagación y de la extensión para NS2 del modelo Durkin, la cual nos permite simular escenarios en espacios tridimensionales y obtener un análisis experimental de los algoritmos más realista. En la sección 5.3 se presenta una serie de dificultades observadas durante el análisis de los algoritmos de aproximación vistos en los capítulos 3 y 4. En la sección 5.4 se detallan algunas de las propuestas para mejorar el desempeño de los algoritmos estudiados bajo el enfoque de una MANET y en la sección 5.5 se presenta el algoritmo propuesto que incluye las mejoras.

5.1 Entorno de simulación NS2

NS2 es una herramienta de simulación de eventos discretos de código abierto diseñado específicamente para la investigación en redes de computadoras. Sus principales usos son: la implementación de nuevos protocolos, la comparación de diferentes protocolos bajo una serie de métricas y la evaluación del tráfico. En general, NS2 proporciona una manera de especificar protocolos de red y simular sus comportamientos correspondientes. Una de las principales ventajas de este simulador es su implementación de los efectos de la capa física, la cual es muy realista y por tanto nos permite tener resultados de las simulaciones más cercanos a la realidad. Este simulador es capaz de trabajar con un gran número de aplicaciones, modelos de tráfico, tipos y elementos de red.

NS2 consiste de dos lenguajes: C++ y una herramienta de lenguaje de comandos orientada a objetos (*OTcl*, *Object-oriented Tool Command*). C++ define los mecanismos internos de la

simulación (como es la implementación detallada de los protocolos) y OTcl establece la simulación mediante la configuración de los objetos así como la programación de los eventos discretos (configura las simulaciones).

5.1.1. Redes ad hoc móviles en NS2

Debido a la ausencia de enlaces de comunicación físicos alambrados, todos los nodos de una red inalámbrica ad hoc son capaces de moverse libremente durante la simulación. NS2 incorpora la comunicación inalámbrica y la movilidad de los nodos en los nodos regulares y define un nuevo tipo de nodos llamados *nodos móviles*.

Los nodos móviles son fundamentales para la simulación, estos procesan y transmiten los paquetes, lo que los convierte en uno de los componentes más importantes para NS2 y para el desarrollo de los algoritmos.

Proceso de configuración de nodos móviles

El proceso de creación de nodos móviles consiste en dos pasos:

1. Configuración del nodo móvil. En este paso se almacena la información de configuración usando el comando:

   ```
   ns node-config -<option> <value>
   ```

   En la Tabla 5.1 se muestran las opciones disponibles y su respectivo valor. Una vez configurados los nodos, el segundo paso consiste en crear los nodos móviles, en lugar de nodos regulares.

2. Construcción de un nodo móvil. En este paso, se crea un nodo móvil usando la sentencia

   ```
   ns node
   ```

Opción	Descripción
`-adhocRouting`	Tipo de protocolo (ej. `AODV`)
`-llType`	Tipo de capa de enlace (ej. `LL`)
`-macType`	Tipo de control de acceso al medio (ej. `Mac/802_11`)
`-ifqType`	Tipo de orden de administración del buffer (ej. `Queue/DropTail/PriQueue`)
`-ifqLen`	Tamaño del buffer en paquetes (ej. cinco paquetes)
`-antType`	Tipo de antena (ej. `Antenna/OmniAntenna`)
`-propType`	Tipo de modelo de propagación de radio (ej. `Propagation/Durkins`)
`-phyType`	Tipo de interfaz de red (ej. `Phy/WirelessPhy`)
`-channelType`	Tipo de canal (ej. `Channel/WirelessChannel`)
`-topoInstance`	Una instancia OTcl que identifica la topología (ej. `new Topology`)
`-agentTrace`	Traza a nivel de agente encendida o apagada (`ON` u `OFF`)
`-routerTrace`	Traza a nivel de enrutamiento encendida o apagada (`ON` u `OFF`)
`-macTrace`	Traza a nivel MAC encendida o apagada (`ON` u `OFF`)
`-movementTrace`	Traza de movimiento encendida o apagada (`ON` u `OFF`)

Tabla 5.1: Detalles de la configuración de un nodo móvil.

5.1.2. Generadores de movilidad y de tráfico

Para llevar a cabo las simulaciones de redes ad hoc móviles en NS2, se escribe un script en OTcl que configura la simulación y se crean dos archivos: un archivo con la descripción del movimiento de los nodos y uno que describe el tráfico o patrón de comunicación en la red. Estos archivos se incluyen en el script OTcl que define la simulación junto con la especificación de las características de los nodos. El resultado de la simulación es un archivo conocido como *traza* donde quedan registrados todos los eventos que ocurrieron durante la simulación.

Generador de movilidad

Existen dos formas para agregar movimiento a los nodos, la primera es que el usuario introduzca de manera manual la posición inicial y final del nodo, y la segunda forma es especificar el movimiento de manera aleatoria. Para especificar de manera aleatoria el movimiento, el modelo más utilizado es el *Random Waypoint*, que se caracteriza por manejar un *tiempo de pausa*.

Bajo el modelo *Random Waypoint*, al inicio de la simulación todos los nodos comienzan en una

posición inicial aleatoria dentro del área de simulación y permanecen estacionarios por un tiempo de pausa, después cada nodo selecciona una posición destino aleatoria y se mueve hacia el destino a una velocidad aleatoria uniformemente distribuida. Una vez en la posición destino, el nodo espera otro tiempo de pausa, selecciona aleatoriamente otro destino y se mueve hacia él, esto se repite hasta que la simulación termina. Se escogió este modelo para las simulaciones desarrolladas en esta tesis.

NS2 provee el generador de movilidad *setdet* que crea sentencias de movilidad deterministas. Esta escrito en C++ y su ejecutable se encuentra en el directorio

~ns/indep-utils/cmu-sen-gen/setdest , donde *~ns* es el directorio donde se instaló el simulador. Existen dos versiones, la utilizada en esta tesis es la versión 2 desarrollada en la Universidad de Michigan.

El comando utilizado para generar el archivo con las sentencias de movilidad es el siguiente:

```
≫ ./setdest -v <version> -n <num_nodes> -t <sim_time> -s <speed_type> -m <min_speed> -M <max_speed> -P <pause_type> -p <pause_time> -x <max_x> -y <max_y>}
```

Donde:

<version> es la versión del generador de movimiento, en nuestro caso toma el valor 2.

<num_nodes> es el número de nodos móviles en la simulación.

<sim_time> es el tiempo de simulación.

<speed_type> es el tipo de velocidad empleada por los nodos, 1 para uniforme.

<min_speed> es la velocidad mínima en metros/segundos.

<max_speed> es la velocidad máxima en metros/segundos.

<pause_type> es el tipo de pausa

<pause_time> es el tiempo de pausa en segundos.

<max_x> dimensión *x* del terreno en que se mueven los nodos, en metros.

<max_y> dimensión y del terreno en que se mueven los nodos, en metros.

Generador de tráfico

NS2 también proporciona otra utilidad independiente *cbrgen.tcl* escrita en Tcl, para crear sentencias OTcl relacionadas al tráfico. A diferencia del generador de movilidad, *cbrgen.tcl* es un script que necesita ser invocado por el interprete *ns* de la siguiente manera:

```
≫ ns cbrgen.tcl -type <cbr lline tcp> -nn <num_nodes> -seed <seed> -mc <max_conn> -r <rate>
```

Donde:

`<cbr lline tcp>` es el tipo de tráfico.

`<num_nodes>` es el número de nodos móviles en la simulación.

`<seed>` es una semilla aleatoria.

`<max_conn>` es el máximo número de conexiones que serán generadas.

`<rate>` es la velocidad de transferencia en bps para tráfico cbr.

Con la ayuda de esta herramienta se generó el tráfico tipo cbr (*constant bit rate*) de diseminación de información empleado en las simulaciones para evaluar los algoritmos implementados.

5.1.3. Archivos de traza

En el archivo de traza se registra la información de todos los eventos ocurridos durante el tiempo de simulación, por lo que es muy importante. Ya que a partir de éste se extrae toda la información necesaria para obtener los resultados que nos ayudan a caracterizar los diferentes algoritmos o protocolos. Existen dos formatos, el primero es conocido como el formato antiguo y en éste la información está resumida, el segundo es conocido como el formato nuevo y aunque es más complicado de leer, su ventaja es la amplia información que registra. Para obtener los resultados en esta tesis se utilizo el formato nuevo.

Para activar el formato nuevo, es necesario escribir los siguientes comandos en nuestro script OTcl:

```
ns_ trace-all <ch>
ns_ use-newtrace
```

Cada línea del archivo de traza inalámbrica, generado al activar el formato nuevo, comienza con el tipo de evento, que puede ser: `(s)` *enviado*, `(r)` *recibido*, `(d)` *desechado* o `(f)` *reenviado*. Y enseguida una serie de etiquetas, cada una con su respectivo valor delante de ella. Las etiquetas se clasifican de la siguiente manera:

Información general:

-t Tiempo

Información del nodo (-N}?):

-Ni Identificador del nodo

-Nx Coordenada *x* del nodo

-Ny Coordenada Y del nodo

-Nz Coordenada Z del nodo

-Nl Nivel de la traza: AGT}/RTR}/MAC}

-Nw Razón

-Ne Nivel de energía (por defecto = -1, no seguimiento del nivel de energía)

Información a nivel IP (-I}?):

-Is Fuente (dir. puerto)

-Id Destino (dir. puerto)

-It Tipo de paquete

-Il Tamaño del paquete

-If Identificador del flujo

-Ii Identificador único (del paquete)

-Iv Tiempo de vida (TTl)

Siguiente salto (-H}?):

-Hs Identificador de este nodo

-Hd Identificador del nodo del siguiente salto

Información a nivel MAC (-M}?):

-Ms Dirección Ethernet fuente

-Md Dirección Ethernet destino

-Mt Tipo Ethernet

-Ma Tiempo de transmisión del paquete

Información de aplicación-ARP (-P arp -P}?):

-Ps Dirección IP fuente

-Pd Dirección IP destino

-Pm Dirección MAC fuente

-Pa Dirección MAC destino

-Po "REQUEST" o "REPLY"

Información de aplicación-CBR (-P cbr -P}?)

-Pi Número de secuencia

-Pf Número de veces que el paquete a sido retransmitido

-Po Mínimo número de saltos para llegar al destino indicado

Información de aplicación-TCP (-P tcp -P}?):

-Ps Número de secuencia

-Pa Número de confirmación

-Pf Número de veces que el paquete a sido retransmitido

-Po Mínimo número de saltos para llegar al destino indicado

5.2. Modelo de propagación Durkin

Se utiliza un modelo de propagación para predecir la potencia de la señal recibida de cada paquete. En la capa física de cada nodo inalámbrico, hay un umbral de recepción. Cuando se recibe un paquete, si la potencia de la señal está por debajo del umbral de recepción, se marca

como un error y es desechado por la capa MAC. En NS2 se utilizan principalmente dos modelos: (1)*Free space* y (2)*Two ray ground reception*.

En el primer modelo se asume que hay un único camino que es recto y directo, entre el emisor y el receptor. En un canal de radio señales móviles esto rara vez ocurre, por lo que el modelo de propagación *Free Space* en la mayoría de los casos es inadecuado. El segundo modelo es una variación del modelo *Free-space*, en este modelo se considera que existen dos rayos, uno directo y otro indirecto, entre dos nodos. Se introduce una distancia cruce después de la cual los rayos reflejados interfieren de manera destructiva con el rayo directo y reducen drásticamente la fuerza del campo. Este modelo ofrece resultados más precisos, para largas distancias, que el modelo *Free-Space*.

Sin embargo, la propagación de las radio señales en redes móviles se lleva sobre terrenos irregulares donde la señal pueden sufrir desvanecimientos, atenuaciones, retardos, interferencias o variaciones temporales, algunas de ellas debido a edificios, montañas, árboles, etc. Por lo que, el terreno de simulación debe tener en cuenta estas pérdidas en la ruta de transmisión entre el emisor y el receptor, ya que puede variar de una simple línea de visión a una que esté severamente obstruida.

Con el fin de tener simulaciones más realistas del desempeño de las redes ad hoc en la vida real, se decidió utilizar la extensión presentada en [11] para el simulador NS2, que implementa el modelo de Durkin [10], que es un modelo que incorpora la naturaleza de la propagación sobre terrenos irregulares y las perdidas causadas por los obstáculos en el camino de las radio señales, lo que permite realizar escenarios de simulación más realistas y analizar la forma en el que las características del terreno afectan al rendimiento de las redes ad hoc.

5.3. Dificultades en algoritmos de aproximación para construir conjuntos dominantes conectados

Los diferentes algoritmos vistos en los capítulos 3 y 4 proporcionan heurísticas útiles para construir un conjunto dominante conectado. Sin embargo, hacen suposiciones poco realistas que complican su aplicación en escenarios reales, tales como:

- Suponer que los nodos conocen la información completa de la topología de la red.

- Asumir que el sistema de transmisión en la capa de control de acceso al medio es completamente confiable y por tanto que todos los mensaje son enviados y recibidos correctamente.
- Suponer que todos los nodos saben en qué fase de la ejecución del algoritmo se encuentran.
- No considerar los posibles cambios en la topología de la red durante la fase de construcción del CDC y aún después de ésta.

Por ejemplo, para el primer punto se requiere de algunos nodos centralizados que actualicen constantemente la información de la topología o que cada nodo dé seguimiento a los demás nodos de la red, lo cual por supuesto tiene un costo. Un ejemplo del segundo punto se da cuando un nodo debe ser elegido como líder, pero para que esto ocurra el nodo debe recibir los mensaje de consentimiento implícito o explícito de todos sus vecinos, y si alguno de estos mensajes no llega entonces el proceso podría fallar.

En particular los algoritmos estudiados en [3] y [19], que construyen conjuntos dominantes conectados en gráficas de discos unitarios de manera distribuida y que de aquí en adelante llamaremos el algoritmo de *Kim* y el algoritmo de *Butenko y Ursulenko*, también asumen algunos de los puntos anteriores.

Una de las suposiciones con mayor impacto al implementar estos algoritmos, es asumir que la capa MAC es completamente fiable, ya que en ambos algoritmos existen etapas donde los nodos dependen de recibir todos los mensajes de un conjunto de nodos en específico para disparar algún determinado evento.

Otro problema observado durante su estudio, es no considerar posibles cambios durante la ejecución del algoritmo. Ya que, aún teniendo velocidades sorprendentes en la transmisión de los paquetes, no se deja caer en el caso que durante la construcción del conjunto dominante conectado la topología cambia, ya sea por la movilidad de los nodos o por una batería agotada. Por lo tanto, de manera muy similar al caso anterior, los algoritmos caen en errores que pueden llevar a que la ejecución del algoritmo se detenga en espera de un paquete que no llegará.

Los problemas anteriores observados son el motivo principal de desarrollar un algoritmo que

construya conjuntos dominantes conectados para tratar el problema de la diseminación de información, aplicando algunas técnicas que mejoren el desempeño de los algoritmos ya existentes. El *algoritmo propuesto* se deberá probar en un simulador que realice los efectos de la capa MAC de la manera más realista posible, como lo es NS2, y debe tomar en cuenta los problemas que se presentan en un red ad hoc móvil, como lo es el cambio constante de la topología de la red. Y para hacer su análisis aún más realista, se utiliza el modelo de propagación Durkin en tres dimensiones como ocurre en la realidad.

Para utilizar el modelo Durkin manteniendo válido el factor de aproximación de los algoritmos implementados para su comparación, incluyendo el algoritmo propuesto, es necesario tomar aquellos algoritmos que hagan su análisis para obtener el valor de su factor de aproximación en 3 dimensiones. Por este motivo, los algoritmos de *Kim* y el de *Butenko y Ursulenko* fueron escogidos para su implementación y comparación contra el *algoritmo propuesto*.

Además, también se lleva a cabo la implementación del algoritmo de la técnica de inundación para su comparación.

Técnicas propuestas para mejorar el desempeño de los algoritmos de aproximación para CDC

Uno de los principales problemas en los algoritmos de *Kim* y de *Butenko y Ursulenko*, se presenta cuando algún nodo espera un paquete de control de un conjunto específico de otros nodos, como respuesta a un paquete enviado previamente, y el paquete esperado no llega. Puede ser que el paquete se haya perdido debido a fallas presentados en la capa MAC o por que alguno o varios nodos cambiaron su posición y por lo tanto también la topología cambió.

El problema anterior se presenta en redes alambradas cuando un nodo envía un mensaje y espera la confirmación del mismo. La técnica que resuelve este problema es hacer uso de temporizadores de tal forma que, si el paquete de confirmación no llega antes de que expire el temporizador, el emisor vuelve a enviar el mensaje.

La técnica del uso de temporizadores se propone para solucionar el problema de la espera de un paquete de un conjunto de nodos. Pero se debe tomar en cuenta que una vez que el temporizador expire la decisión de reenviar el paquete y esperar por su respuesta no se debe

manejar al igual que en una red alambrada. Ya que si un nodo cambió su posición o dejó de formar parte de la red ad hoc, éste nunca responderá y por lo tanto el temporizador podría expirar y reiniciarse una y otra vez. Las opciones son entonces, determinar un número de veces que el temporizador se reinicie o, la primera vez que el temporizador expire dar por hecho que el nodo ya no existe y no esperar más por su paquete.

Cuando la topología cambia durante la construcción del CDC el estado del nodo, que cambió su posición o que ya no forma parte de la red, puede afectar la construcción virtual del CDC. Por ejemplo, si en alguna de las etapas de los algoritmos de *Kim* y de *Butenko y Ursulenko* un nodo marcado como parte del MIS o del CDC deja de formar parte de la red, sus nodos hijos dejan de recibir los mensajes de control y los hijos de sus hijos, etc. esto puede llegar a afectar la construcción del CDC, un ejemplo claro es si el nodo unía dos subconjuntos de la red. Por ejemplo, en la Figura 5.1 para los algoritmos de *Kim* y de *Butenko y Ursulenko* si el nodo 6 (con C(6) = 8, 9, 5 y 4) en (a) deja de formar parte de la red, la topología cambia como se muestra en (b). Entonces los nodos hijos 8, 9, 5 y 4 dejan de recibir los paquetes del nodo padre 6. Es claro que el nodo 3 podría continuar con la construcción del CDC, pero no lo hace debido a que en los nodos hijos de 3 no está el nodo 4.

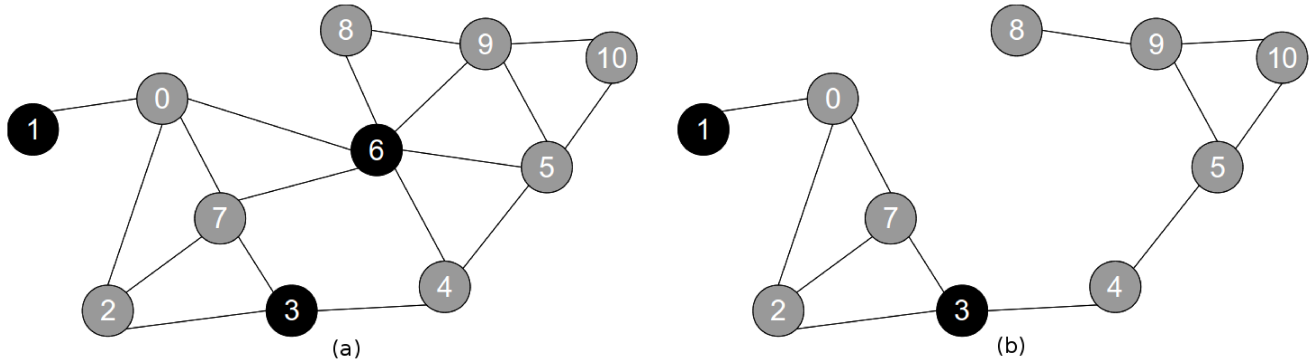

Figura 5.1: La formación del conjunto dominante conectado se ve afectada si un nodo cambia su posición o se desconecta de la gráfica.

Para el caso en general, todo nodo que detecte cambios en la información de su vecindario durante la construcción del CDC, debe tomar medidas dependiendo de su situación. En el ejemplo de la Figura 5.1 el nodo 4, al darse cuenta de que el nodo 6 ya no está, debe colocar al nodo 3 en su lugar y el nodo 3 incluirlo en el conjunto de sus nodos hijos, para que los paquetes lleguen al nodo 4 y continuar la construcción del CDC.

Algunas de las medidas en el *algoritmo propuesto*, tomadas por los nodos al detectar un cambio en la topología por medio de la información proporcionada por sus vecinos, son:

- Si un nodo *x* marcado en blanco detecta que tiene un nuevo vecino *y* negro o azul, se marca en gris y marca al nodo *y* como su padre y el nodo *y* lo marca como su hijo.
- Si un nodo *x* marcado en blanco o gris detecta que todos sus vecinos son grises, entonces se marca de negro si está en la primer etapa del algoritmo o en azul si está en la segunda etapa.
- Si un nodo *x* gris detecta que el nodo marcado como padre ya no está en sus vecinos y existe algún otro nodo *y* negro o azul, marca a *y* como su padre y *y* lo marca como su hijo
- Si un nodo *x* negro detecta que el nodo marcado como su padre ya no está en sus vecinos y sus vecinos siguen siendo todos grises, entonces informa a su vecino con mayor id, para que se encargue de restablecer la relación de padres e hijos y convertirse en su nodo padre.
- Si un nodo *x* marcado en negro detecta que tiene un nuevo vecino *y* azul, se marca en azul (si cubre a por lo menos un nodo) y marca al nodo *y* como su padre y el nodo *y* lo marca como su hijo.

Durante el tiempo transcurrido entre dos ejecuciones del algoritmo, (dependiendo de como se haya programado y que puede depender a su vez de la movilidad de la red) también se presentan cambios en la topología que detectan los nodos a través de sus vecinos. Durante estos lapsos de tiempo es necesario reconstruir el conjunto dominante virtual en caso de que se detecte que se ha desconectado. Un nodo detecta que se ha desconectado del conjunto dominante cuando entre sus vecinos no existe alguno que forme parte del CDC, ya que todo nodo debe tener al menos un vecino en el conjunto dominante, aún formando él parte de éste porque es conectado.

Bajo esta situación, la única alternativa es pedir a alguno de sus vecinos (que sí tenga un vecino que forme parte del conjunto) que forme parte del conjunto dominante para que le puedan llegar los mensajes. Se elige al nodo con el mayor número de identificación, ya que así es más probable que otros nodos también lo sugieran para formar parte del conjunto si se

presenta el caso. En la Figura 5.2 se muestra un ejemplo de esta situación. En (a) se muestra la topología sin cambios, en (b) el nodo 1 y 2 detectan que el nodo 0 ya no está en sus vecinos, el nodo 2 no tiene problemas ya que él sigue teniendo un nodo azul adyacente. Sin embargo el nodo 1 debe pedirle al nodo 2 que se una al CDC para que los mensajes de diseminación le lleguen, en (c) se muestra la topología después de que el nodo 2 se une al CDC. Obsérvese que el nodo 0 puede decidir ya no formar parte del conjunto dominante porque ya no es necesario en él.

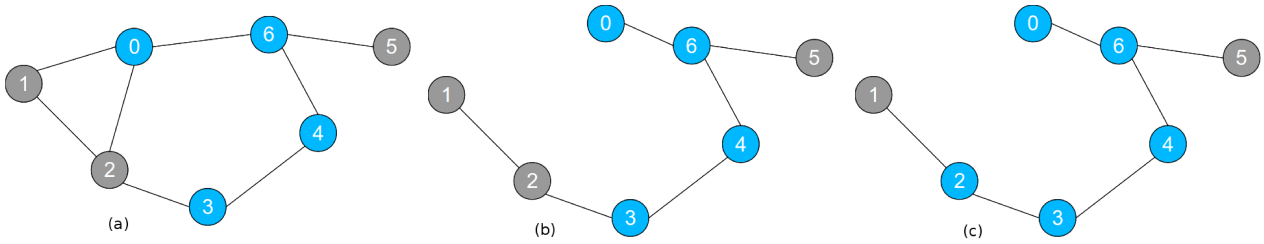

Figura 5.2: El nodo 2 se une al CDS para reconstruir el CDS virtual

Otra mejora que puede afectar el tamaño del conjunto dominante conectado es incluir algunos nodos que ya no son necesarios para obtener el conjunto dominante conectado. Este caso se presenta en los algoritmos de *Kim* y de *Butenko y Ursulenko* cuando se añade un nodo *x* al CDC y enseguida se añaden los nodos negros adyacentes a *x*, que forman parte del MIS. Se puede dar el caso de que un nodo del MIS ya no cubra a algún otro nodo y por lo tanto ya no sea necesario.

En este caso, la técnica consiste en que un nodo pueda decidir si forma parte o no del conjunto dominante conectado, dependiendo de si cubre o no a algún otro nodo y de que cambie su estado para que sea considerado en otra posible iteración. En la Figura 5.3 se presenta un pequeño ejemplo donde al aplicar la técnica anterior el tamaño del CDC se reduce, en (a) se le pide al nodo 0 unirse al CDC, después de que se une al CDC (se colorea de azul) en (b), el nodo negro 1 también debería hacerlo, pero como ya no cubre a ningún otro nodo entonces el nodo 1 decide no unirse, cambia su estado y finalmente el CDC quedaría como en (c).

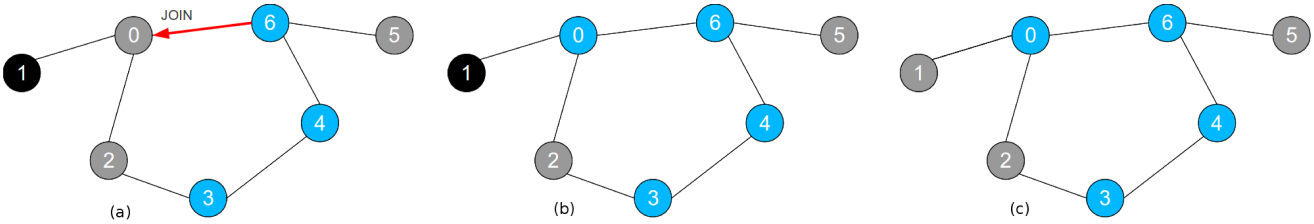

Figura 5.3: Un nodo es capaz de decidir si se une o no al CDC, en base a la información de sus vecinos.

5.5. Algoritmo propuesto

El *algoritmo propuesto*, calcula un conjunto dominante conectado en dos etapas. En la primera etapa calcula un conjunto independiente maximal y en la segunda une los nodos del MIS añadiendo otros nodos. Es un algoritmo distribuido que toma en cuenta los problemas mencionados en la sección 5.3, surgidos en el contexto de las MANETs, e implementa las técnicas propuestas en la sección 5.4. Su objetivo es construir un conjunto de nodos dominantes de manera virtual de tal forma que estos nodos sean los encargados de retransmitir los paquetes de diseminación, y de esta manera reducir el uso de los recursos en la red. El *algoritmo propuesto* está basado en los algoritmos de *Kim* y de *Butenko y Ursulenko* que tienen factores de aproximación de $14.937\ lline\ opt\ lline\ +\ 1.083$ y $22\ \ lline\ opt\ lline$ respectivamente.

Al inicio todos los nodos están marcados en blanco, para un nodo estar marcado en blanco significa que es un nodo que acaba de unirse a la red o que la ejecución del algoritmo está esperando a comenzar. Conforme el algoritmo procede, un nodo se marca en negro si es seleccionado para formar parte del conjunto independiente maximal y sus vecinos se marcan en gris. Un nodo se marca en azul si es seleccionado para formar parte del conjunto dominante conectado. Antes de que un nodo negro sea marcado en azul, el nodo decide si se une o no al CDC de acuerdo a la información de sus vecinos, es decir si no cubre a ningún nodo no se marca de azul. Además, de manera periódica en un nodo se ejecutan las rutinas de la sección 5.4 que restablecen los caminos para continuar con la construcción virtual del CDC, en caso de ser necesario.

Al final del algoritmo, el conjunto de nodos marcados en azul forma parte del conjunto dominante conectado, los nodos en este conjunto son los encargados de retransmitir los paquetes de diseminación. Al igual que durante la construcción del CDC, durante el lapso entre las ejecuciones del algoritmo, se ejecutan rutinas de reparación del CDC para que un nodo que se ha desconectado del CDC vuelva a conectarse y de esta manera reciba los paquetes de diseminación.

Parte 1: Algoritmo propuesto (inicio)

```
1  Todo nodo x establece C(x) ← N(x)
2  Se selecciona un nodo r con el grado máximo como la raiz
3  Color(r) ← negro
4  foreach (x ∈ C(r)) do
5  |  Color(x) ← gris
6  |  P(x) ← r
7  |  C(x) ← C(x) \ N[r]
8  end
9  r envía un mensaje ASK
10 r inicia temporizador TIMER_1
```

Parte 2: Algoritmo propuesto (rutina de un nodo x al recibir un mensaje)

```
1  switch Tipo de mensaje do
2  |  case DELETE
3  |  |  if (x ≠ P(nodo_emisor)) then
4  |  |  |  C(x) ← C(x) \ {nodo_emisor}
5  |  |  end
6  |  end
7  |  case IGNORE
8  |  |  C(x) ← {y|Color(y) = gris}
9  |  end
10 |  case FIN
11 |  |  if (Color(x) = azul) then
12 |  |  |  x reenvía mensaje FIN
13 |  |  end
14 |  |  x inicia temporizador TEMP_3
15 |  end
16 |  case COLOR
17 |  |  if (Etapa de ejecución ≠ 1 AND Etapa de ejecución ≠ 2) then
18 |  |  |  Color(x) ← azul
19 |  |  end
20 |  end
21 end
```

Parte 3: Algoritmo propuesto (rutina de un nodo x al recibir un mensaje)

```
1  switch Tipo de mensaje do
2      case ASK
3          if (Color(x) ≠ blanco) then
4              x reenvía el mensaje ASK a todo nodo en C(x)
5              x inicia temporizador TIMER_1
6          end
7          else
8              x envía un mensaje REP(x, m(x)) de regreso a r, donde
               m(x) es el número de nodos blancos en N[x]
9          end
10     end
11     case REP(y,m(y))
12         if (Color(x) ≠ blanco) then
13             if (m(y) > m(x)) then
14                 Index(x) ← y
15                 m(x) ← m(y)
16             end
17         end
18     end
19     case JOIN
20         if (Color(x) ≠ blanco) then
21             x reenvía el mensaje JOIN al nodo en Index(x)
22         end
23         else
24             Color(x) ← negro
25             P(x) ← emisor del mensaje JOIN
26             x envía un mensaje DELETE
27             foreach (y ∈ C(x)) do
28                 Color(y) ← gris
29                 P(y) ← x
30                 y envía un mensaje DELETE
31             end
32         end
33     end
34 end
```

Parte 4: Algoritmo propuesto (rutina de un nodo x al recibir un mensaje)

```
1  switch Tipo de mensaje do
2      case ASK_
3          if (Color(x) = azul) then
4              x reenvía mensaje ASK_
5              x inicia temporizador TIMER_2
6          end
7          else
8              x envía mensaje REP_(x,b(x)) de regreso a r, donde b(x)
               es el número de nodos negros en N[x]
9          end
10     end
11     case REP_(y, b(y))
12         if (Color(x) = azul) then
13             if (b(y) > b(x)) then
14                 Index(x) ← y
15                 b(x) ← b(y)
16             end
17         end
18     end
19     case JOIN_
20         if (Color(x) = azul) then
21             x reenvía el mensaje JOIN_ al nodo en Index(x)
22         end
23         else
24             Color(x) ← azul
25             P(x) ← emisor del mensaje JOIN_
26             x envía mensaje DELETE
27             foreach (nodo negro y ∈ N(x)) do
28                 if (y cubre a por lo menos un nodo de la red) then
29                     Color(y) ← azul
30                     P(y) ← x
31                     y envía mensaje DELETE
32                 end
33             end
34         end
35     end
36 end
```

Parte 5: Algoritmo propuesto (rutina de un nodo x al expirar un temporizador)

```
1  switch Tipo de temporizador do
2      case TIMER_1
3          if (x ≠ r) then
4              x envía mensaje REP(x, m(x)) al nodo en P(x)
5          end
6          else
7              if (m(r) = 0) then
8                  r envía mensaje IGNORE, finaliza la primer etapa
                   del algoritmo, el conjunto de nodos negros forma un
                   MISColor(r) ← azul  foreach (y ∈ C(r)) do
9                      P(y) ← r
10                     C(y) ← C(y) \ N[r]
11                 end
12                 r envía mensaje ASK_r inicia temporizador TIMER_2
13             end
14             else
15                 r envía mensaje JOIN al nodo en Index(r)
16             end
17         end
18     end
19 end
```

Parte 6: Algoritmo propuesto (rutina de un nodo x al expirar un temporizador)

```
1  switch Tipo de temporizador do
2      case TIMER_2
3          if (x ≠ r) then
4              x envía mensaje REP(x, b(x)) al nodo en P(x)
5          end
6          else
7              if (b(r) = 0) then
8                  r transmite mensaje FIN a través del CDC, finaliza
                   la segunda etapa del algoritmo, el conjunto de nodos
                   azules forma un CDC
9              end
10             else
11                 r envía mensaje JOIN_ al nodo en Index(r)
12             end
13         end
14     end
15     case TIMER_3
16         if (∀y ∈ C(x), Color(y) ≠ azul) then
17             x envía mensaje COLOR a uno de los nodos en C(x)x
               reinicia temporizador TEMP_3
18         end
19     end
20 end
```

5.6. Análisis del factor de aproximación

El algoritmo presentado conserva el valor del factor de aproximación calculado en [19]. El factor de aproximación se obtiene a partir de dos cotas superiores. La primera cota es para el número de nodos independientes en una esfera unitaria y la segunda para el número de nodos que se añaden para conectar el conjunto independiente maximal en la segunda etapa del algoritmo.

5.6.1. Cota superior del conjunto independiente maximal

La primera cota, es una cota superior mejorada para el número de vértices en un conjunto independiente maximal en términos del número de vértices en un conjunto dominante conectado óptimo. Para obtener el valor de la cota se considera cuantos nodos independientes puede contener una esfera unitaria, se establece una cota superior para el número de nodos independientes en dos esferas unitarias adyacentes y finalmente se demuestra, en base a lo anterior, la cota superior para el número de vértices en el conjunto independiente maximal.

- Cota superior para el número de nodos independientes en una esfera unitaria.

El valor de esta cota se basa en el problema acerca del *número de ósculos* de Gregory-Newton [33]. El número de ósculos $k(S_3)$, es el número de esferas unitarias independientes que pueden tocar simultáneamente la superficie de una esfera unitaria y en [16] se muestra que $k(S_3) = 12$.

Lema 2 *El número de ósculos* $k(S_3) = 12$

Del lema anterior se deduce el siguiente lema:

Lema 3 *Para cualquier vértice u en una gráfica de esferas unitarias G, el vecindario N_G(u) contiene a lo más 12 vértices independientes.*

- Cota superior para el número de nodos independientes en dos esferas unitarias adyacentes. Del lema 3, hay a los más 24 nodos independientes en la unión de dos esferas unitarias, usando la fórmula de Euler se mejora esta cota superior a 22. La idea básica de la prueba es similar a la de derivar una cota superior para el problema del número de ósculos en [33], donde primero se proyecta cada nodo independiente v_i, con

i = 1, 2, ..., t, sobre la superficie de la unión de dos esferas unitarias ($B_1 \cup B_2$, con centros u_1 y u_2), si $v_i \in B_1 \setminus B_2$ (respectivamente $B_2 \setminus B_1$). Luego se une cada punto P_i (punto de intersección del radial, que va de u_1 o u_2 hacia v_i , con la superficie de la esfera) sobre la superficie de la unión de las dos esferas para dividirla en piezas. Cada pieza es una \textit{cara}. Dibujando adecuadamente las curvas, se puede obtener una cota inferior para las áreas de cada cara, luego usando la formula de Euler $t + f - e = 2$, donde *t, f* y *e* son el número de vértices, caras y aristas respectivamente, se obtiene una cota superior para el número de vértices en la unión de dos esferas unitarias. Por otra parte, ya que los vértice $v_i s$ son independientes, podemos ver que para cualquier par de enteros distintos *i* y *j*, $d(P_i, P_j) > 1$, es decir, $P_i s$ son también independientes.

Para dividir la superficie de $B_1 \cup B_2$ se establece una curva entre cualquiera dos puntos P_i y P_j cuya distancia este entre 1 y $3\arccos(1/7)/\pi$. La intersección de B_1 y B_2 es un círculo denotado por L. Para establecer una curva entre P_i y P_j se tiene dos casos:

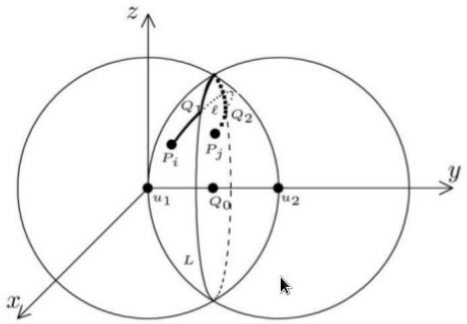

Figura 5.4: Unión de dos puntos sobre la superficie de la misma esfera.

- Caso 1. Si P_i y P_j se encuentran sobre la superficie de la misma esfera, se unen mediante un arco geodésico *l*. Si *l* no se encuentra totalmente sobre la superficie de B_1 y B_2 , entonces *l* cruza *l* en dos puntos Q_1 y Q_2 como se muestra en la Figura 5.4. Entonces la curva que une P_i y P_j se compone de la unión de tres arcos $l_{Pi, Q1}$, $l_{Q1, Q2}$ y $l_{Q2, Pj}$.

- Caso 2. Si P_i y P_j se encuentran sobre las superficies de diferentes esferas unitarias, sea Π el plano que forman los puntos P_i , P_j y Q_0 , donde Q_0 es el punto medio

del segmento de linea entre u_1 y u_2 . Entonces Π cruza con el círculo l en dos puntos, sea Q el que está más cerca a P_i y P_j . Unimos P_i y P_j a *Q* mediante un arco geodésico $l_{Pi,\ Q}$ y $l_{Q,\ Pj}$ como se muestra en la Figura 5.5. La curva que une P_i y P_j esta compuesta de la unión de $l_{Pi,\ Q}$ y $l_{Q,\ Pj}$. Las curvas dibujadas no se cruzan entre si.

Ahora se obtiene una cota inferior para el área de las caras formadas sobre la superficie de las esferas. Una k-cara es una cara *S* acotada por *k* curvas y *A(S)* el área de la cara *S*. Sea A_k el área mínima de una k-cara regular (una cara que esta completamente sobre la superficie de una esfera unitaria) y $\tilde{A}_k$ el área mínima de una cara irregular (una cara que esta sobre la superficie de dos esferas). Una *k*-cara cuyo limite esta acotado por *l* como en el Caso 1 se considera como una cara irregular.

Lema 4 *Para caras regulares de la gráfica obtenida anteriormente,* $A_3=0.5512$ *...,* $A_4=1.3338$ *...,* $A_5=2.2261$ *... En general,*

$$A_k \geq (k-2)A_3 \qquad (5.1)$$

Para caras irregulares, $\tilde{A}_3=0.4076$ *...,* $\tilde{A}_4=0.9949$ *...,* $\tilde{A}_5=1.8732$ *.... En general,*

$$\tilde{A}_k \geq (k-2)\tilde{A}_3 \quad \textit{para k = 3, 4, 5, ...} \qquad (5.2)$$

Los valores para las áreas $A_k s$ se encuentran en [33] y también las posiciones para las *k*- caras y los valores para las áreas $\tilde{A}_k s$ en [19].

Lema 5 *El número de nodos independientes en* $(B_1 \cup B_2)\backslash(B_1 \cap B_2)$ *es a lo más de 20.*

La prueba de este Lema es equivalente a mostrar que $t \leq 20$. Sea f_i el número de *i*-caras (incluyendo caras regulares e irregulares), y $\tilde{f}_i$ el número de *i*-caras irregulares que pasan por *L*.

De la formula de Euler y el teorema de Handshaking [25],

$$\begin{aligned} 2\mathrm{t}-4 &= 2e-2\mathrm{f} \\ &= 3\mathrm{f}_3+4\mathrm{f}_4+5\mathrm{f}_5+\ldots-2(f_3+f_4+f_5+\ldots) \qquad (5.3) \\ &= f_3+2\mathrm{f}_4+3\mathrm{f}_5+\ldots \ . \end{aligned}$$

Del lema 4 y de 5.3, tenemos que:

$$6\pi \geq A_3(f_3-\tilde{f}_3)+\tilde{A}_3\tilde{f}_3+A_4(f_4-\tilde{f}_4)+\tilde{A}_4\tilde{f}_4+A_5(f_5-\tilde{f}_5)+\tilde{A}_5\tilde{f}_5+\ldots$$

$$\geq A_3(f_3+2\mathrm{f}_4+3\mathrm{f}_5+\ldots)-(A_3-\tilde{A}_3)(\tilde{f}_3+2\tilde{f}_4+3\tilde{f}_5+\ldots)$$

$$= A_3(2\mathrm{t}-4)-(A_3-\tilde{A}_3)(\tilde{f}_3+2\tilde{f}_4+\ldots). \qquad (5.4)$$

donde 6π es una cota superior para el área de la superficie de $(B_1 \cup B_2)$. La ecuación significa que si se encuentra una cota superior de $\tilde{f}_3+2\tilde{f}_4+\ldots$, se puede conseguir el máximo valor de *t*.

Primero se asume que cada cara irregular es una *3*-cara. Sean $S_1=P_1P_2P_3$ y $S_2=P_2P_3P_4$ dos *3*-caras talque el arco común P_2P_3 pasa por L . Entonces la proyección de $S_1 \cup S_2$ sobre *l* tiene longitud de arco de al menos 0.8744 [19]. Ya que la longitud de arco de *l* es $\sqrt{3\pi}$, y $\sqrt{3\pi}/0.8744=6.223$, existen a lo más seis pares de *3*-caras. Por lo tanto, $\tilde{f}_3 \leq 13$ y sustituyendo en 5.4 se obtiene que $t \leq 20.792$. Ya que *t* es un entero, se tiene que $t \leq 20$. En general cualquier *i*-cara, con $i>3$, se puede dividir en algunas *j*-caras regulares con área de al menos A_j y algunas *3*-caras irregulares con área de al menos $\tilde{A}_3$. Además, la proyección de una nueva *3*-cara irregular así obtenida sobre *l* ocupa una longitud de arco no menor que la que una original *3*-cara irregular lo hace. Por lo tanto, por un argumento similar al anterior, tenemos que $t \leq 20$ [19].

Lema 6 *El número de nodos independientes en la unión de dos esferas unitarias adyacentes es a lo más 22.*

Sean B_1 , B_2 dos esferas unitarias adyacentes. Supongamos que $I=\{v_{1,}\ldots,v_s\}$ es el conjunto de nodos independientes en $B_1 \cup B_2$. Si $B_1 \cap B_2$ contiene al menos dos $v_i s$, entonces, por el Lema 3, $s \leq 2\times 12-2=22$ y ya está. Por lo tanto, supongamos que $B_1 \cap B_2$ contiene a lo más un $v_i s$. De acuerdo con el Lema 5, $(B_1 \cup B_2)\backslash(B_1 \cap B_2)$ contiene a lo más $20\,v_i s$ y $s \leq 20+1=21$.

- El siguiente resultado es una cota superior para el número de nodos independientes en una gráfica de esferas unitarias en términos de un conjunto dominante conectado óptimo.

Teorema 7 *Sea M un conjunto independiente maximal de una gráfica de esferas unitarias*

G. Entonces $|M| \leq 10.917\,OPT_{CDS} + 1.083$ *, donde* OPT_{CDS} *es el número de vértices en un conjunto dominante conectado óptimo de G.*

Sea *C* un conjunto dominante conectado de *G* con $|C| = OPT_{CDS}$ y *G[C]* la subgráfica de *G* formada por *C*. Del Lema 2 se deduce que *G[C]* tiene un árbol de expansión mínimo *T* con grado máximo de a lo más 12. Se muestra que hay a lo más $10.917|T| + 1.083$ nodos independientes en el vecindario de *T* por inducción en $|T|$. Cuando $|T| = 1\,o\,2$, la afirmación es verdadera por los Lemas 3 y 6. Supongamos que $|T| \geq 3$. Como *T* es un árbol existe un nodo interno *v* que es adyacente a máximo un nodo interno.

Sea *u* el vecino interno de *v* o la raíz de *T*. Supongamos que $x_{1,}\ldots, x_k (k \leq 11)$ es el conjunto de vecinos hojas de *v*. Entonces, $T' = T - \{v, x_{1,}\ldots, x_k\}$ es un árbol de expansión mínimo de $G[C - \{v, x_{1,}\ldots, x_k\}]$ que es una gráfica de esferas unitarias formada por todos los nodos en $C - \{v, x_{1,}\ldots, x_k\}$. Además el grado máximo de *T*' es 12 ya que, como *T* es un árbol de expansión mínimo con grado máximo de 12, *T*' también es un árbol de expansión de $G[C - \{v, x_{1,}\ldots, x_k\}]$ con grado máximo de 12. Supongamos que *T*' no es un árbol de expansión mínimo. Entonces debe haber otro árbol de expansión mínimo *T*''. Ahora, $T'' + \{uv, vx_{1,}\ldots, vx_k\}$ es un árbol de expansión de *G[C]* con peso menor a *T*, lo cual contradice la hipótesis de *T* es un árbol de expansión mínimo.

Por hipótesis inductiva, hay a lo más $10.917(|T| - k - 1) + 1.083$ nodos independientes en el vecindario de $|T| - \{v, x_{1,}\ldots, x_k\}$. Por otra parte, por el Lema 3, para cualquier nodo $x_i (1 \leq i \leq k)$, hay a lo más 11 nodos independientes en su vecindario, también independientes de *v* y por el Lema 6 el vecindario de *v* y x_k contienen a lo más 21 nodos independientes también independientes de *u*. Por lo tanto, hay a lo más

$$10.917(|T| - k - 1) + 1.083 + 21 + 11(k - 1)$$

$$= 10.917|T| + 1.083 + 0.083\text{k} - 0.917$$

$$\leq 10.917|T| + 1.083$$

nodos independientes en el vecindario de *T*.

5.6.2. Cota superior del número de nodos añadidos para conectar el conjunto independiente maximal

Una vez que se tiene la primera cota superior para el número de nodos independientes en términos del conjunto dominante conectado óptimo, se calcula la segunda cota para el número de nodos añadidos que conectan el conjunto independiente maximal calculado, también en términos del conjunto dominante conectado óptimo.

Lema 7 *Para cualquier nodo u ∈ G, N(u) contiene a lo más 12 nodos independientes.*

Ya que G es una gráfica de esferas unitarias, lo anterior es cierto por el Lema 3.

Lema 8 *Después de que la primera etapa del algoritmo finaliza,* $|C \setminus M_0| \leq 4.02|OPT_{CDS}|$ *, donde* OPT_{CDS} *es un conjunto dominante óptimo de G.*

Para probar el lema anterior, supongamos que $OPT_{CDS}=\{x_1, x_2, \ldots, x_t\}$, que es un conjunto dominante óptimo de *G*. Sea

- $S_i=\{x_i\} \cup (N(x_i) \setminus OPT_{CDS})$ para $i = 1$, y
- $S_i=\{x_i\} \cup ((N(x_i) \setminus OPT_{CDS}) \setminus \cup_{j=1}^{i-1} S_j)$ para $i <= t$.

Entonces, $\{S_1, S_2, \ldots, S_t\}$ forma una partición de *V(G)*.

Consideremos el siguiente sistema de ponderación. Es decir, cuando se selecciona un nodo *v* y se añade $\{v\} \cup M_{v,C}$ a *C*, a cada nodo del conjunto independiente maximal *x* en $M_{v,C}$ se le asigna un peso $w(x)=1/|M v, C|$. Obsérvese que si *C* no es un conjunto independiente maximal, entonces, tenemos que $|M_{v,C}|>0$. Además,

$$\sum_{\forall S_i} \sum_{x \in M_0 \cap S_i} w(x)$$

es la cota superior de $|C \setminus M_0|$ del algoritmo.

Ahora, se muestra que para cada *i*, $\sum_{x \in M_0 \cap S_i} w(x) \leq 4.02$. Se denota $a_j=(M_0 \cap S_i) \setminus C_j$, donde C_j es el conjunto *C* después de la *j*-ésima iteración, y $C_0=\{r\}$ es el conjunto inicial. Supongamos que j_e es el primer índice de la iteración después de la cual todos los nodos de $M_0 \cap S_i$ se incluyen en *C*. Por lo tanto, $a_j>0$ para $j=0,1,\ldots,j_e-1$ y $a_{j_e}=0$. Para simplificar la declaración, se asume $a_{j-1}-a_j>0$ para $j=1,2,\ldots,j_e$. Después de la primera

iteración, el número de nodos del conjunto independiente maximal que reciben pesos es a_0-a_1 y el peso asignado a cada nodo es a lo más $1/(a_0-a_1)$. Ya que $a_0-a_1>0$, algunos nodos del conjunto independiente maximal en $M_0\cap S_i$ se añaden a *C* en la primera iteración. Por lo tanto, antes de la *j*-ésima iteración para $j\in\{2,3,\ldots,j_e\}$, x_i es adyacente a *C*. Mediante la estrategia voraz, vemos que $|M_{v_j,C_{j-1}}|\geq|M_{x_i,C_{j-1}}|$, donde v_j es el nodo elegido, que no esta en el conjunto independiente maximal, en la *j*-ésima iteración por el algoritmo. Después de la *j*-ésima iteración, $a_{j-1}-a_j$ nodos de $M_0\cap S_i$ reciben pesos y el peso asignado a cada nodo es $1/|M_{v_j,C_{j-1}}|\leq 1/|M_{x_i,C\,j-1}|=1/a_{j-1}$. Por lo tanto,

$$\sum_{x\in M_0\cap S_i} w(x)\leq\frac{1}{a_0-a_1}(a_0-a_1)+\sum_{j=2}^{j_e}\frac{1}{a_{j-1}}(a_{j-1}+a_j) \quad .$$

Como se asumió que $a_{j-1}-a_j>0$ para cualquier $j=1,2,\ldots,j_e$, el número de nodos del conjunto independiente maximal en $(M_0\cap S_i)\backslash C_{j-1}$ estrictamente disminuye en cada iteración. Por lo tanto, por el Lema 7, $j_e\leq 12$ y el segundo término a la derecha de la expresión está acotada por *H(11)*, donde *H* es la función armónica. Por lo tanto, tenemos que

$$\sum_{x\in M_0\cap S_i} w(x)\leq 1+H(11)\leq 4.02 \quad .$$

Teorema 8 *El tamaño del conjunto de nodos C generado por el algoritmo no es de más de* $14.937|OPT_{CDS}|+1.083$ *, donde* OPT_{CDS} *es un conjunto dominante conectado óptimo de G y* $|OPT_{CDS}|=t$.

Por el Teorema 7, $|M_0|\leq 10.917|OPT_{CDS}|+1.083$, y el Lema 8, el algoritmo añade a lo más $4.02|OPT_{CDS}|$ para conectar los nodos en M_0 . Por lo tanto,

$$\begin{aligned}|C| &= \sum_{i=1}^{t}\sum_{x\in M_0\cap S_i} w(x)+|M_0| \\ &\leq 4.02t+10.917|OPT_{CDS}|+1.083 \\ &\leq 14.937|OPT_{CDS}|+1.083\end{aligned}$$

Capítulo 6

Análisis experimental

Este capítulo comienza con una descripción en la sección 6.1 de las métricas utilizadas para evaluar los resultados de las simulaciones para poder así describir el desempeño de los algoritmos de diseminación de información bajo el contexto de las MANETs 3D. En la sección 6.2 se presentan los diferentes escenarios considerados en las simulaciones. Por último, en la sección 6.3 se muestran los resultados obtenidos de manera gráfica.

6.1. Métricas para la evaluación del desempeño

Para comparar el desempeño del algoritmo propuesto para el cálculo de conjuntos dominantes conexos se usaron las siguientes métricas de comparación del desempeño de un protocolo.

6.1.1. Relación de entrega de paquetes

La relación de entrega de paquetes (*packet delivery ratio*), corresponde al valor porcentual de paquetes exitosamente recibidos por los nodos destino. Se calcula dividiendo el número de paquetes recibidos entre el número de paquetes que debieron ser recibidos. Esta métrica nos indica que tan eficaz es el algoritmo o protocolo.

6.1.2. Retardo de extremo a extremo

El retardo de extremo a extremo (*end to end delay*), se define como la media de la diferencia de tiempo transcurrida entre el instante en que se generan los paquetes y el momento en que llegan a su destino.

Por lo tanto, el retardo de extremo a extremo incluye todos los retrasos de la red, tal como el

tiempo de transmisión, los retrasos provocados por las actividades del tráfico, el espacio en colas. Por ejemplo, puede ocurrir que los paquetes tomen mucho tiempo en llegar a su destino debido a largas colas en algunos o en todos los nodos intermedios por los que pueda pasar el paquete.

6.1.3. Sobrecarga total

La sobrecarga total (*total overhead*), es el número total de paquetes enviados, es decir, es la suma de la sobrecarga de control y la sobrecarga de datos. Esta métrica nos indica la eficiencia del protocolo.

Sobrecarga de control

La sobrecarga de control (*control overhead*), es el número total de paquetes de control enviados.

Ya que la cantidad de tráfico de control incrementa conforme la red crece, la sobrecarga de control se vuelve una importante medida de la escalabilidad del protocolo y por lo tanto de la red.

Sobrecarga de datos

La sobrecarga de datos (*data overhead*), es el número total de paquetes de datos enviados.

6.2. Escenarios de simulación

Para llevar a cabo la simulación de los algoritmos se realizaron diferentes tipos de escenarios. En el primer tipo de escenario, cuyos parámetros y su respectivo valor se muestran en la Tabla 6.1, se varía el número de nodos y el área de simulación, como se muestra en la siguiente lista, manteniendo así la densidad con el fin de medir la escalabilidad de la red, es decir, medir el desempeño a medida que la red se vuelve cada vez más grande.

- 25 nodos, área de simulación de 750m $\times$ 750m $\times$ 200m
- 50 nodos, área de simulación de 1000m $\times$ 1000m $\times$ 200m
- 100 nodos, área de simulación de 1500m $\times$ 1500m $\times$ 200m

Parámetro	Valor
Número de nodos	[25, 50, 100]
Área de simulación	[750m × 750m × 200m, 1000m × 1000m × 200m, 1500m × 1500m × 200m]
Tiempo de simulación	100s
Posición inicial de los nodos	Aleatoria
Rango de transmisión	250m
Modelo de propagación	Durkin
Tipo de tráfico	cbr

Tabla 6.1: Escenario nodos estáticos.

Para el segundo tipo de escenario se fijó el número de nodos y el área de simulación, en este caso lo que se varía es el tiempo de pausa de los nodos. El objetivo en este tipo de escenario es evaluar el desempeño de la red a medida que los nodos son más movedizos. Los parámetros y su respectivo valor se muestran en la Tabla 6.2.

Para cada uno de los experimentos se deja un tiempo de la simulación para que los algoritmos que calculan el conjunto dominante, incluyendo el algoritmo propuesto, converjan. Luego, en lo que resta del tiempo de simulación, en cada segundo se toma una fuente al azar para que genere paquetes de diseminación. En el caso de los escenarios con nodos dinámicos, para cada experimento el algoritmo que calcula el conjunto dominante se inicia de manera periódica.

Parámetro	Valor
Número de nodos	50
Área de simulación	1000m × 1000m × 200m
Tiempo de simulación	100s
Posición inicial de los nodos	Aleatoria
Rango de transmisión	250m
Modelo de propagación	Durkin
Modelo de movilidad	Random Waypoint
Tiempo de pausa	[0s, 3s, 9s, 27s]
Velocidad mínima	1m/s
Velocidad máxima	20m/s
Tipo de tráfico	cbr

Tabla 6.2: Escenario nodos dinámicos.

En el tercer tipo de

escenario se varia el número de paquetes de diseminación por segundo. El objetivo es observar mejor los efectos adversos de la fuerza bruta usada en el algoritmo de inundación. Los parámetros y su respectivo valor se muestran en la Tabla 6.3.

Para este experimento se deja un tiempo de la simulación para que el *algoritmo propuesto* converja. Luego, en lo que resta del tiempo de simulación, en cada segundo se toman respectivamente dos, cuatro, ocho, dieciséis y treinta y dos fuentes al azar para que generen paquetes de diseminación.

Parámetro	**Valor**
Número de nodos	50
Área de simulación	1000m × 1000m × 200m
Tiempo de simulación	100s
Posición inicial de los nodos	Aleatoria
Rango de transmisión	250m
Modelo de propagación	Durkin
Modelo de movilidad	Random Waypoint
Tiempo de pausa	9s
Velocidad mínima	1m/s
Velocidad máxima	20m/s
Paquetes de diseminación por segundo	[2, 4, 8, 16, 32]
Tipo de tráfico	cbr

Tabla 6.3: Escenario con variación en el número de paquetes de diseminación.

6.3. Resultados de la simulación

A continuación se presentan las diferentes gráficas obtenidas de la información extraída de los archivos de trazas, generados por el simulador a partir de los escenarios mostrados en la sección anterior.

6.3.1 Escenarios con nodos estáticos

Relación de entrega de paquetes

La gráficas mostradas para la métrica *relación de entrega de paquetes* son de las más importantes, ya que nos hablan de la calidad de conjunto dominante construido por cada uno de los algoritmos en comparación. Es decir, entre mayor sea la cantidad de paquetes entregados, significa que el conjunto dominante construido cubre la mayor cantidad de nodos. Aunque también se debe tomar en cuenta que debido a las pérdidas de paquetes a causa de las diferentes razones bajo el contexto de las MANETs esta medida puede verse afectada.

Lo que nos dice la gráfica de la Figura 6.1 es que para el *algoritmo propuesto* se tiene un porcentaje de alrededor del 95% de paquetes entregados sin variar mucho a medida que la red crece. En comparación de los algoritmos de *Kim* y de *Butenko y Ursulenko*, en los que el desempeño decae a medida que se aumenta el número de nodos, el algoritmo tiene una mejora indudable ya que logra mantenerse casi constante en su relación de entrega de paquetes.

Observamos también que el algoritmo de *inundación* mostró un desempeño mejor que el *algoritmo propuesto*, lo cual es un resultado esperado debido a su alta redundancia en el cubrimiento de un nodo, este hecho se ve reflejado como negativo en las gráficas de sobrecarga de paquetes de datos.

El intervalo de confianza con probabilidad del 90% afirma el mejor desempeño respecto a los algoritmos de *Kim* y de *Butenko y Ursulenko*, exceptuando para el caso con menor número de nodos, que es de 25, ya que los intervalos tienen un cruce lo que indica que en este caso los resultados pueden llegar a ser iguales, es decir, que pueden tener un porcentaje con el mismo valor de paquetes entregados.

Retardo de extremo a extremo

Para observar el comportamiento de esta métrica fue necesario reportarla en más de una gráfica, una para cada distinto número de nodos en la red. El objetivo es ver y comparar el retardo según el número de saltos que el paquete dio para llegar a su destino. Esto es debido a que, como se observo en la gráfica de la Figura 6.1 que muestra la *relación de entrega de paquetes*, los algoritmos de *Kim* y de *Butenko y Ursulenko* tienen un porcentaje menor en paquetes entregados, lo que implica que muchos de los paquetes dan solo unos pocos saltos ya que si diesen todos los saltos correspondientes los paquetes llegarían a todos los nodos. Bajo este hecho el retardo de los paquetes, ya que dan un menor número de saltos, es menor a que

si diesen más saltos, por lo cual para obtener una adecuada comparación es necesario reportar el retardo según el número de saltos que dieron los paquetes.

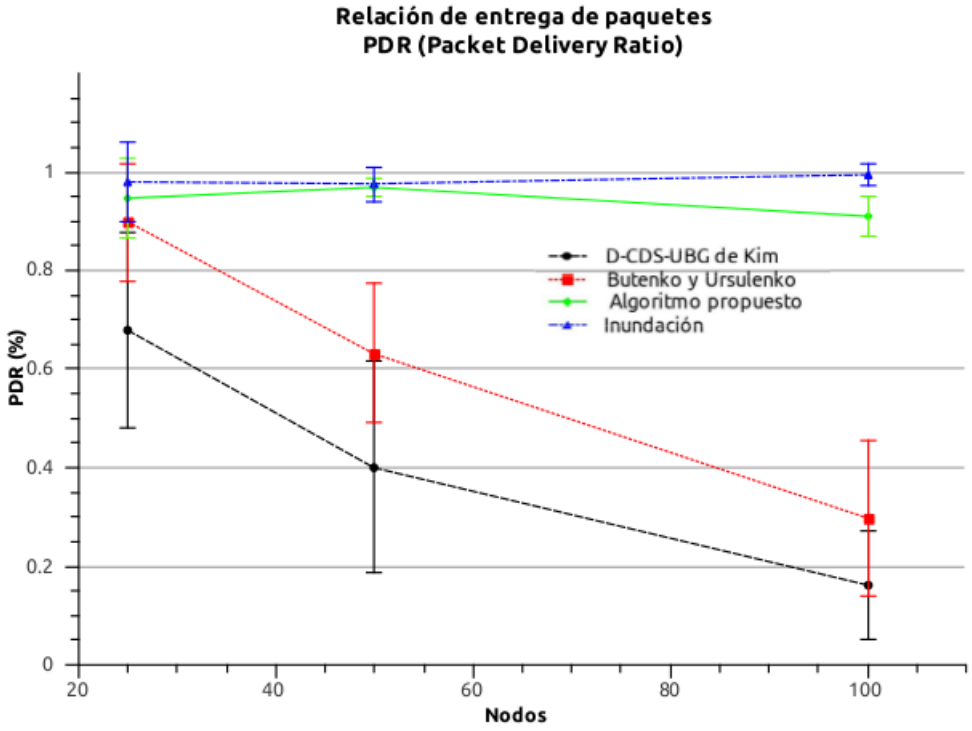

Figura 6.1: Relación de entrega de paquetes en escenarios estáticos.

Las gráficas de las Figuras 6.2, 6.3 y 6.4 para 25, 50 y 100 nodos respectivamente, muestran un comportamiento similar en cuanto al retardo para los algoritmos de *Kim*, de *Butenko y Ursulenko* y el *algoritmo propuesto*.

El algoritmo de *inundación*, igualmente para todas los casos, muestra un retardo menor. Este hecho también es esperado ya que, al retransmitir todos los nodos, se espera que un nodo reciba por primera vez un paquete proveniente del camino con menor costo, ya sea por distancia o por tráfico de la red y por lo tanto con menor retardo.

También, las gráficas de las Figuras 6.3 y 6.4 nos permiten reafirmar que los paquetes entregados en los algoritmos de *Kim* y de *Butenko y Ursulenko* dan un menor número de saltos en comparación con el *algoritmo propuesto*.

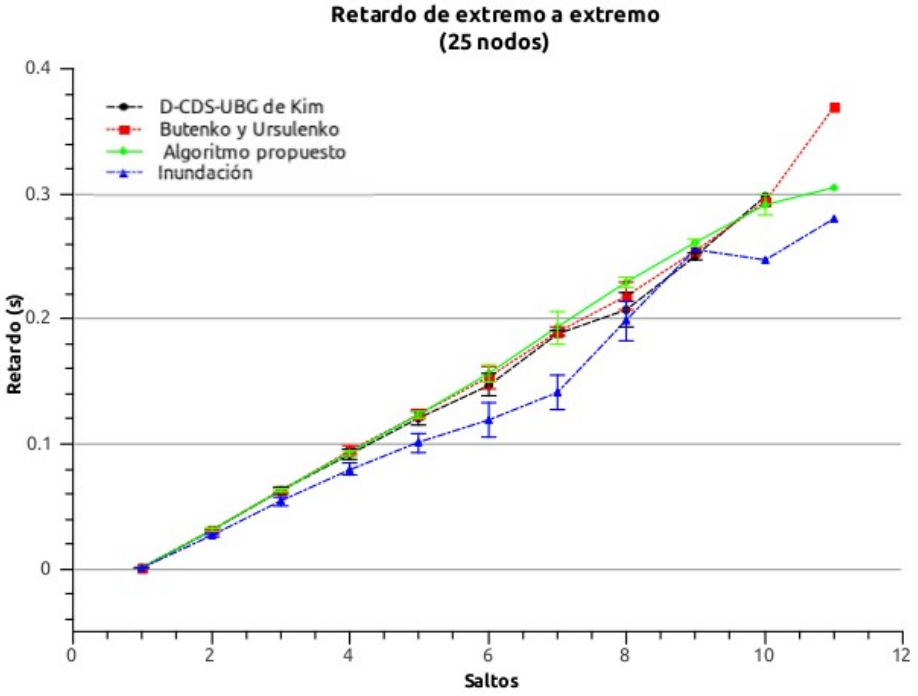

Figura 6.2: Retardo de extremo a extremo en escenarios estáticos de 25 nodos.

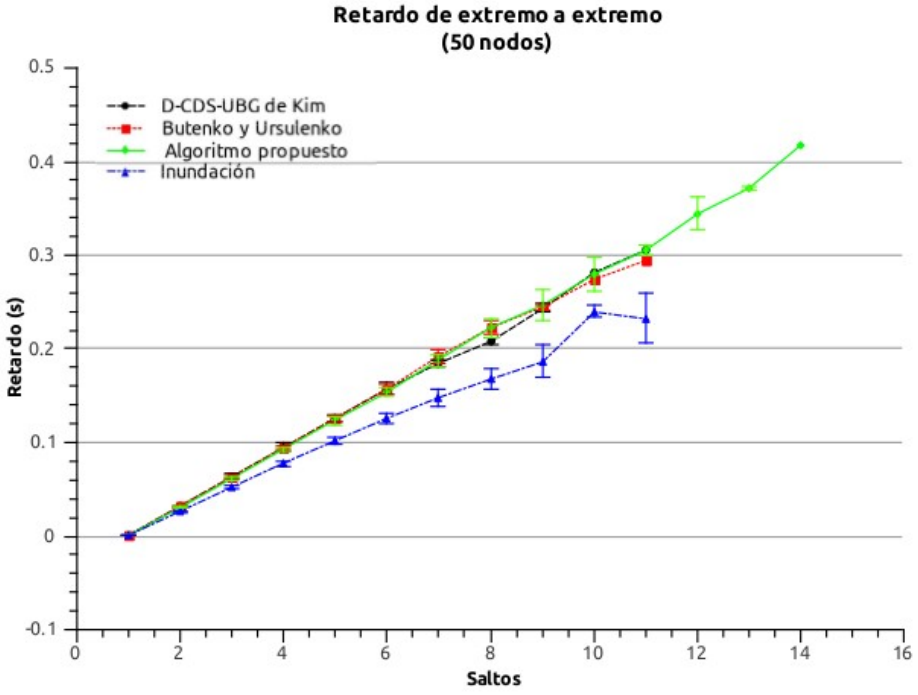

Figura 6.3: Retardo de extremo a extremo en escenarios estáticos de 50 nodos.

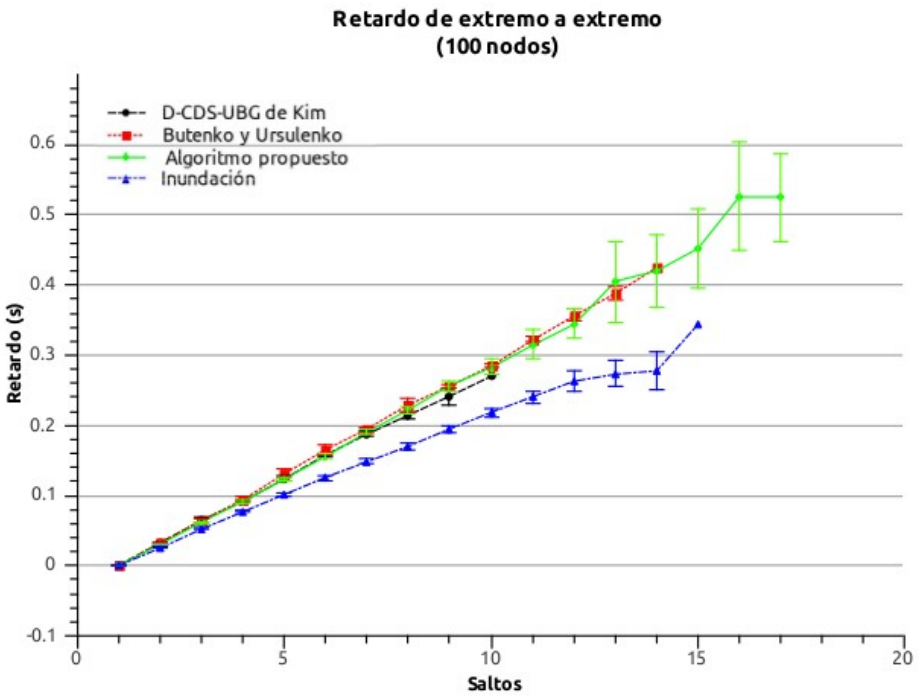

Figura 6.4: Retardo de extremo a extremo en escenarios estáticos de 100 nodos.

Sobrecarga total

Como ya se ha mencionado, los algoritmos de *Kim*, y de *Butenko y Ursulenko* muestran en la gráfica de la Figura 6.1 un porcentaje de entrega de paquetes que disminuye a medida que la red crece, lo cual es desfavorable. Lo anterior nos permite obviar que la sobrecarga de paquetes, tanto de datos como de control, será menor que la sobrecarga de paquetes del *algoritmo propuesto*. Las gráficas de las Figuras 6.5, 6.7 y 6.6 afirman que el *algoritmo propuesto* tiene una mayor sobrecarga de paquetes en comparación con los algoritmos de *Kim* y de *Butenko y Ursulenko*.

Otro resultado de suma importancia es la diferencia de sobrecarga de paquetes entre el algoritmo de *inundación* y el *algoritmo propuesto*, la cual es menor en el *algoritmo propuesto*. Ya que es en estas gráficas, que muestran la sobrecarga de paquetes, donde se nota el hecho de que el uso de la técnica de inundación resulta ser costosa debido al *número de transmisiones de paquetes* que realiza. Y es por este motivo que decidimos usar algoritmos de aproximación que disminuyan el costo que tiene llevar a cabo la técnica de inundación, con el fin de mejorar el desempeño de las MANETs bajo el problema de la diseminación de información.

En la gráfica de la Figura 6.5 se muestra la sobrecarga total de paquetes. En el caso del algoritmo de *inundación* se trata únicamente de paquetes de datos, ya que esta técnica no requiere de ningún paquete de control, y en el caso del *algoritmo propuesto* lo que se refleja es la suma de los paquetes de datos y los paquetes de control, estos últimos necesarios para formar el conjunto dominante y mantener la conectividad virtual de este conjunto.

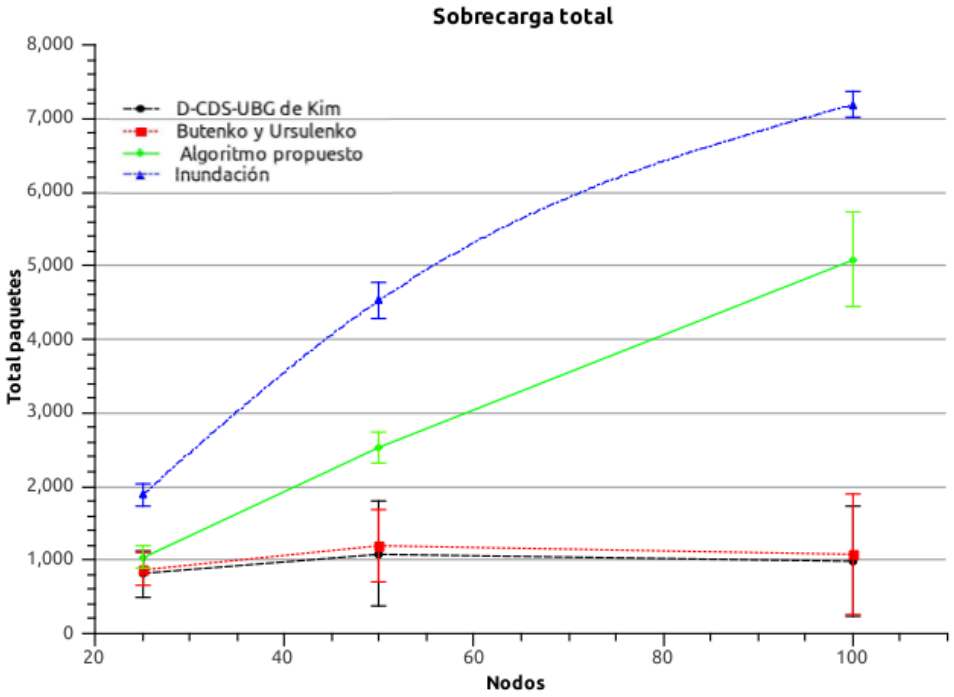

Figura 6.5: Sobrecarga total en escenarios estáticos.

- **Sobrecarga de datos**

  La gráfica de la Figura 6.6 muestra la gran diferencia en el número de retransmisiones de paquetes de diseminación entre la técnica de inundación y algún algoritmo de aproximación más eficiente para el mismo objetivo, diseminar información. En este caso el *algoritmo propuesto* sería el algoritmo eficiente ya que, con una mucho menor cantidad de paquetes de datos, logra un porcentaje de alrededor del 95% de entrega de paquetes.

- **Sobrecarga de control**

  La gráfica de la Figura 6.7 muestra la cantidad de paquetes de control utilizados por cada uno de los algoritmos comparados. Sin embargo, después de considerar que los

algoritmos de *Kim* y de *Butenko y Ursulenko* tienen un bajo porcentaje de entrega de paquetes, nos damos cuenta que el resultado mostrado en la gráfica no refleja el número de paquetes real para formar un conjunto dominante por cada uno de estos algoritmos. Finalmente, tomando en cuenta este hecho, la comparación en esta gráfica pierde sentido.

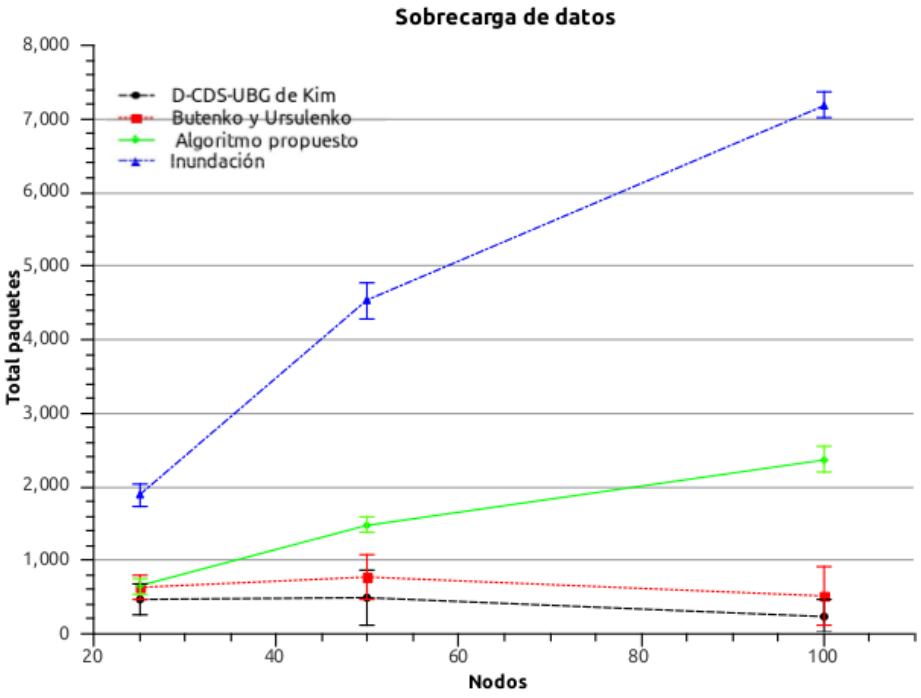

Figura 6.6: Sobrecarga de datos en escenarios estáticos.

6.3.2. Escenarios con nodos dinámicos

Relación de entrega de paquetes

Al igual que en la gráfica de la Figura 6.1, que muestra los resultados a medida que la red crece, la gráfica de la Figura 6.8 en escenarios con nodos móviles muestra un desempeño mejor para el *algoritmo propuesto* en cuanto a porcentaje de paquetes entregados, con una baja de alrededor del 10% de paquetes entregados cuando los nodos son completamente móviles, esto es, cuando el tiempo de pausa es cero, que es el peor de los casos en cuanto a movilidad. Para los demás casos, en cuanto a tiempo de pausa, el algoritmo se mantiene casi constante en su desempeño. Y con respecto al algoritmo de *inundación* sucede lo mismo que para los

escenarios estáticos, se observa que tiene un mejor desempeño debido a su alta redundancia en el cubrimiento de un nodo, hecho que se ve reflejado como negativo en las gráficas que muestran la sobrecarga de datos.

El intervalo de confianza con probabilidad del 90% afirma el mejor desempeño respecto a los algoritmos de *Kim* y de *Butenko y Ursulenko*.

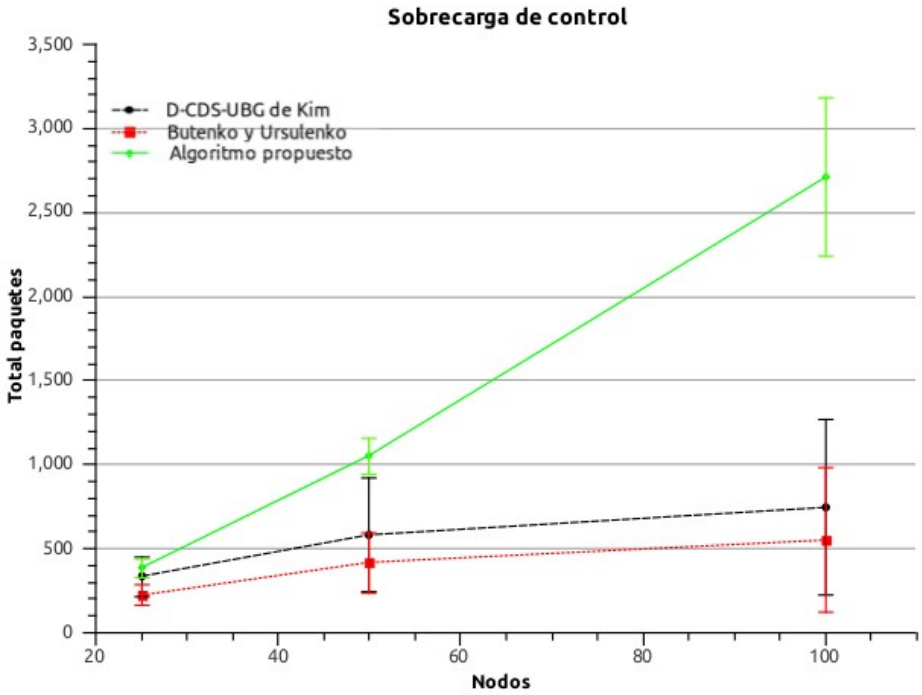

Figura 6.7: Sobrecarga de control en escenarios estáticos.

Retardo de extremo a extremo

Las gráficas de las Figuras 6.9, 6.10, 6.11 y 6.12 tiene un comportamiento similar a las gráficas que muestran los resultados, para esta misma métrica, de los escenarios estáticos. En ellas se puede ver que los algoritmos de *Kim*, de *Butenko y Ursulenko* y el *algoritmo propuesto* muestran un retardo que no es muy variable en cada número de saltos. También en las gráficas de las Figuras 6.10, 6.11 y 6.12 se reafirma que los paquetes de los algoritmos de *Kim* y de *Butenko y Ursulenko*, dan un menor número de saltos y por tanto obtienen un menor porcentaje de entrega de paquetes.

También se observa que el algoritmo de *inundación*, muestra un retardo menor, ya que un nodo recibe por primera vez un paquete proveniente del camino con menor costo y por lo

tanto con menor retardo.

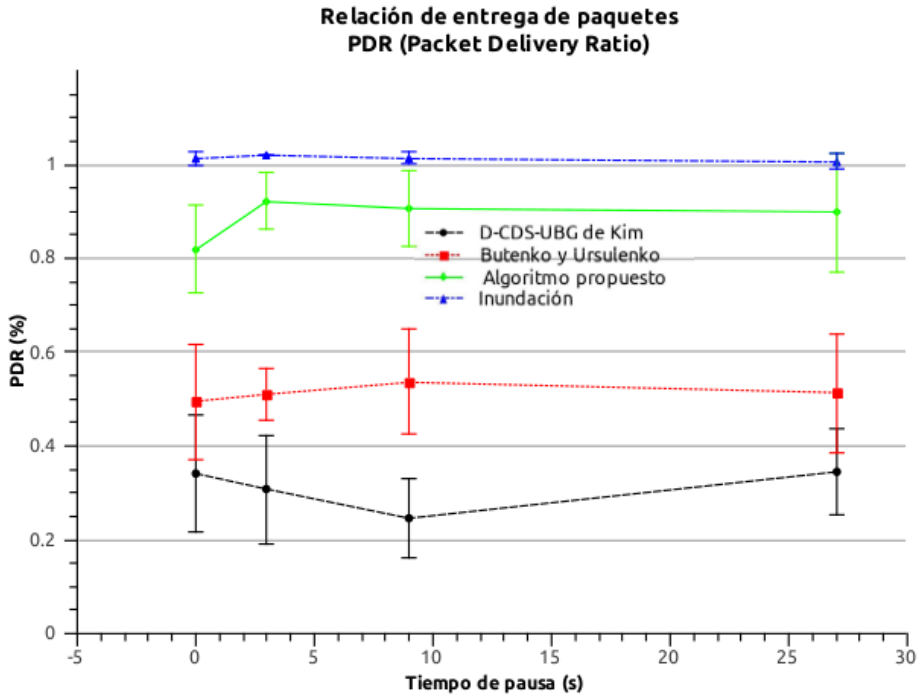

Figura 6.8: Relación de entrega de paquetes en escenarios dinámicos.

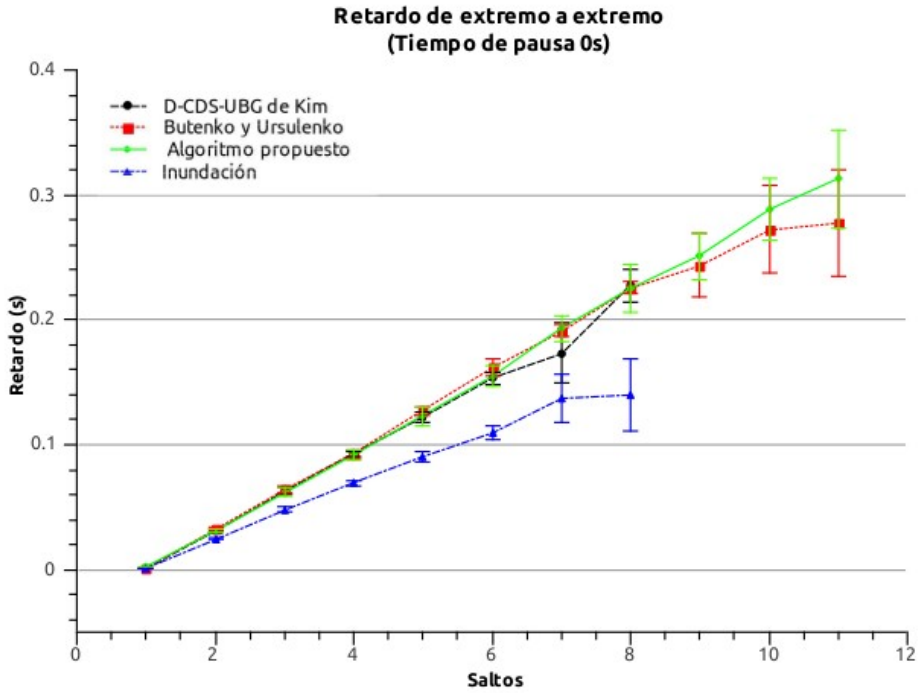

Figura 6.9: Retardo de extremo a extremo en escenarios dinámicos (pausa 0s).

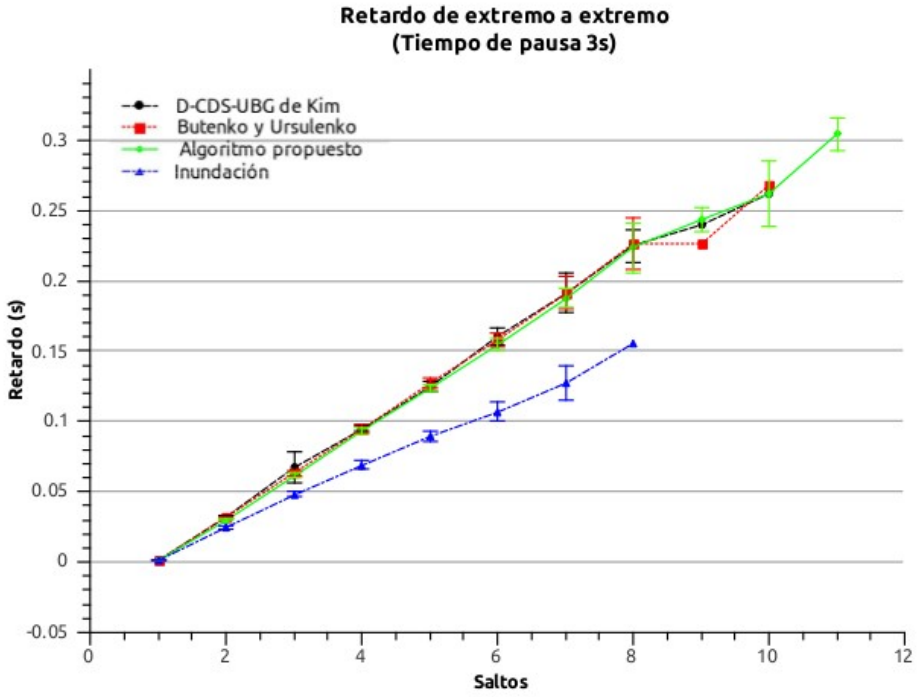

Figura 6.10: Retardo de extremo a extremo en escenarios dinámicos (pausa 3s).

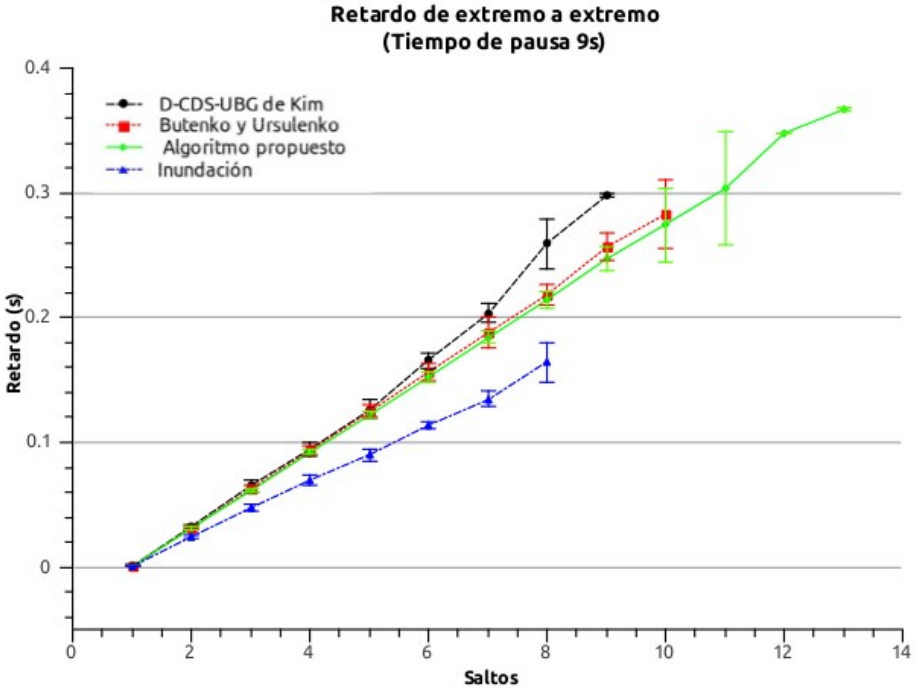

Figura 6.11: Retardo de extremo a extremo en escenarios dinámicos (pausa 9s).

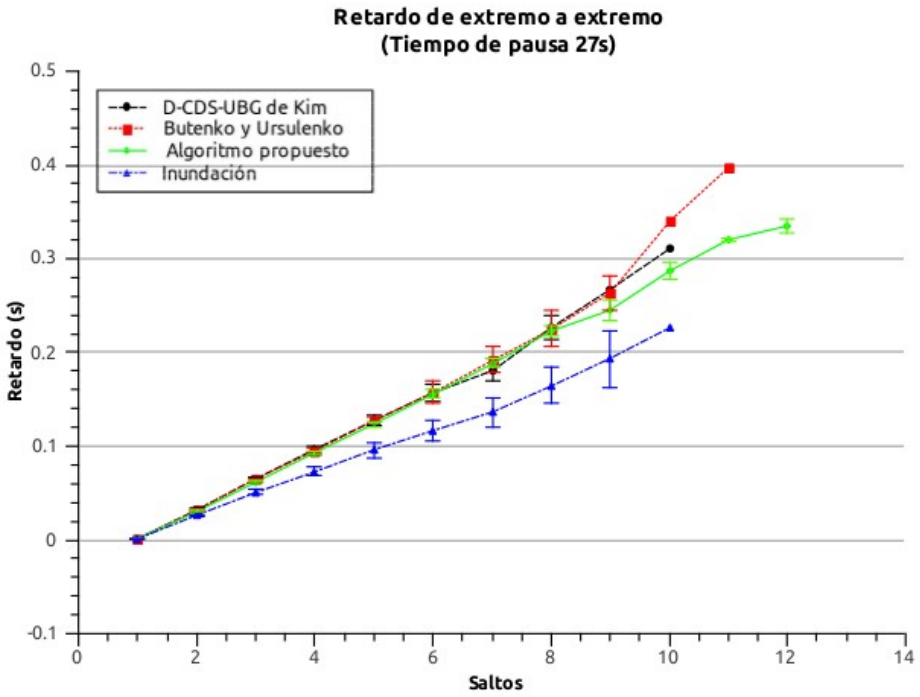

Figura 6.12: Retardo de extremo a extremo en escenarios dinámicos (pausa 27s).

Sobrecarga total

La sobrecarga total mostrada en la gráfica de la Figura 6.13, no nos permite obviar que debido al bajo porcentaje de paquetes entregados (resultado mostrado en la gráfica de la Figura 6.8) por los algoritmos de *Kim* y de *Butenko y Ursulenko*, deberíamos tener un sobrecarga de paquetes mayor por el *algoritmo propuesto*. Esto se cumple para las simulaciones del algoritmo de *Butenko y Ursulenko*, pero no respecto al intervalo de confianza calculado con probabilidad del 90%, lo que implica que los resultados podrían llegar a ser iguales.

Respecto al algoritmo de *inundación*, éste sigue teniendo una mayor sobrecarga en comparación a los demás algoritmos. A pesar de ser únicamente paquetes de datos lo que refleja, a diferencia de los otros algoritmos que, además de los paquetes de datos, también incluyen los paquetes de control necesarios para formar el conjunto dominante de manera periódica.

También podemos observar (Figura 6.13) que, para el tiempo de pausa igual a cero, tenemos una baja en la sobrecarga de paquetes para el *algoritmo propuesto*. Esta baja proviene principalmente de la sobrecarga de datos mostrada en la gráfica de la Figura 6.14. Este

resultado es esperado debido al comportamiento de la gráfica de la Figura 6.8, para el mismo tiempo de pausa, ya que al disminuir el porcentaje de entrega de paquetes se espera que haya una baja en la sobrecarga de paquetes.

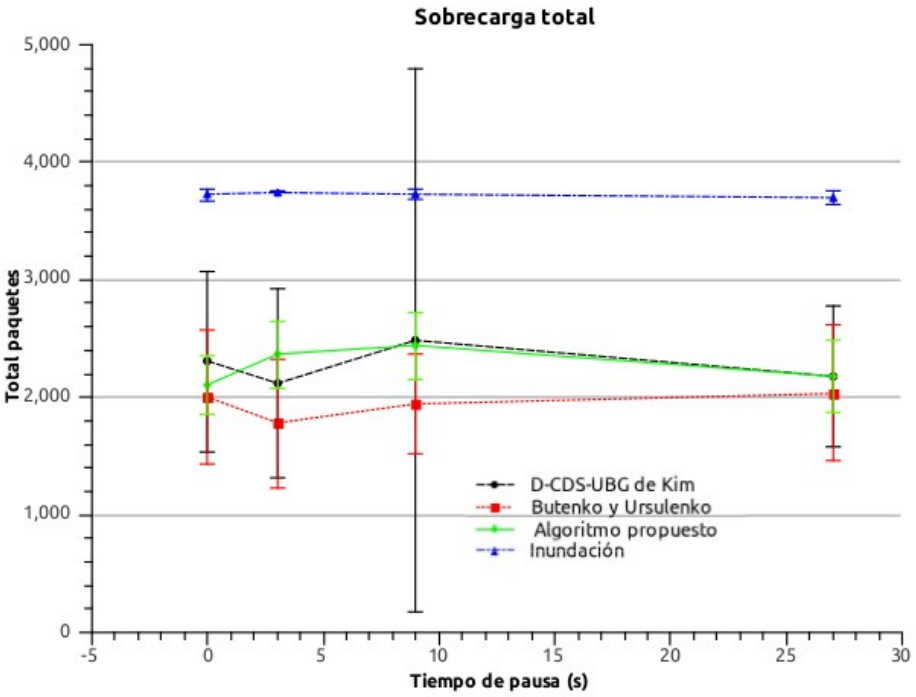

Figura 6.13: Sobrecarga total en escenarios dinámicos.

- **Sobrecarga de datos**

  La gráfica de la Figura 6.14 muestra, de manera muy similar a la gráfica de la Figura 6.6, la gran diferencia en el número de paquetes de datos transmitidos en la técnica de inundación y los transmitidos por algún algoritmo de aproximación eficiente. En este caso el *algoritmo propuesto* resulta más eficiente al mantener casi constante la sobrecarga de paquetes de datos, y muy por debajo del algoritmo de *inundación*. Los algoritmos de *Kim* y de *Butenko y Ursulenko* no resultan muy eficientes debido a su bajo porcentaje de entrega de paquetes, como ya se ha mencionado.

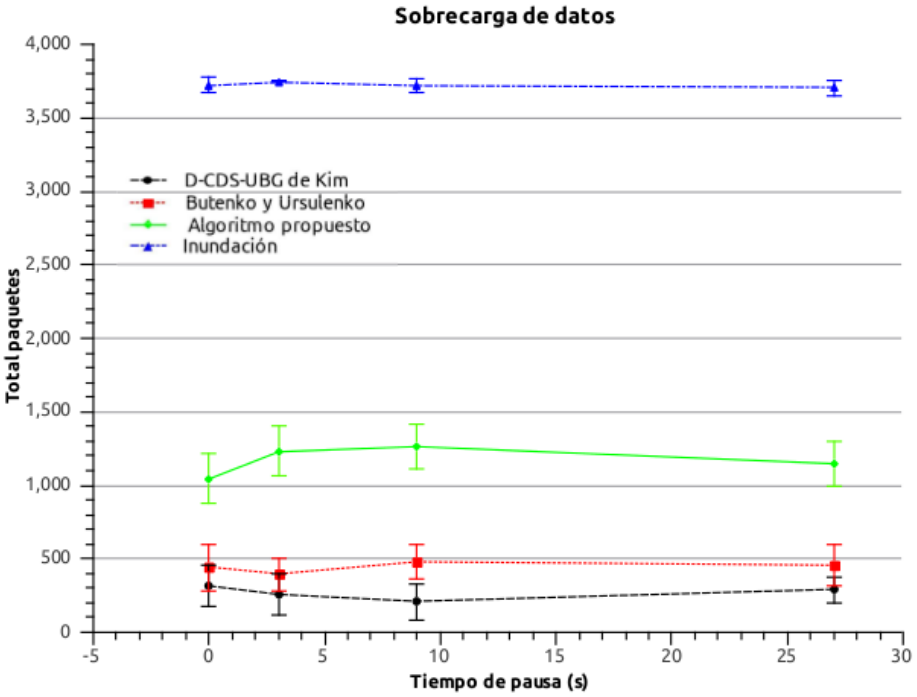

Figura 6.14: Sobrecarga de datos en escenarios dinámicos.

- **Sobrecarga de control**

  En la gráfica final de la Figura 6.15, tenemos un comportamiento no obvio. Este comportamiento se justifica debido a las técnicas implementadas por el *algoritmo propuesto* para mantener la conexión virtual del conjunto dominante un tiempo mayor entre las periódicas ejecuciones del algoritmo. Lo anterior significa que el algoritmo mantendrá el conjunto dominante conectado por más tiempo y por lo tanto son menos las veces que este requiere ejecutarse en comparación con los algoritmos de *Kim* y de *Butenko y Ursulenko*. Como se ejecuta menos veces, hace uso de una cantidad menor de paquetes de control.

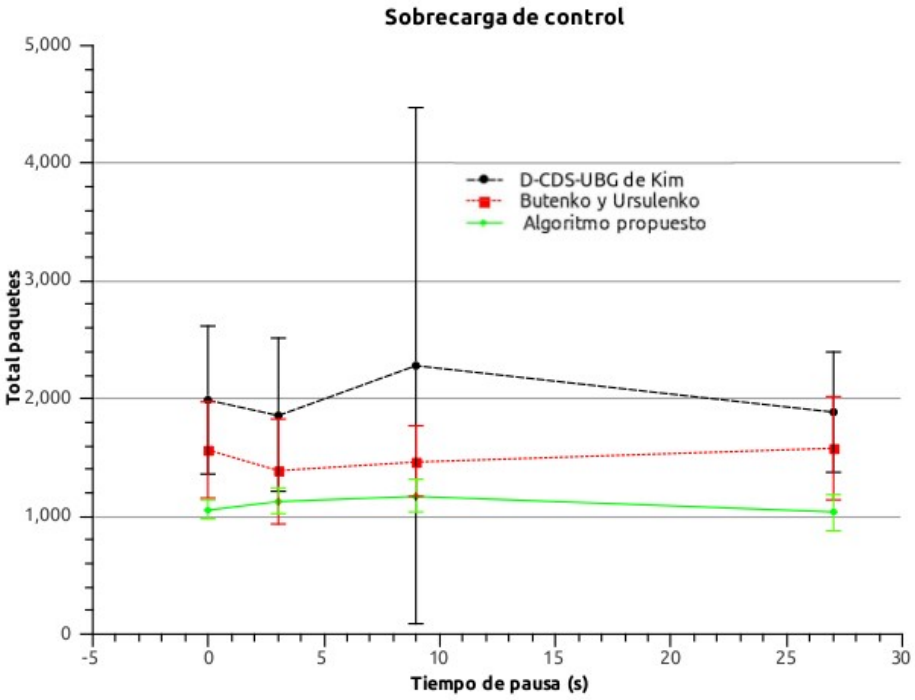

Figura 6.15: Sobrecarga de control en escenarios dinámicos.

6.3.3. Escenarios con variación en el número de paquetes de diseminación

Relación de entrega de paquetes

La gráfica mostrada en la Figura 6.16 para la métrica *relación de entrega de paquetes* nos muestra que el porcentaje de paquetes exitosamente entregados baja conforme el número de paquetes de diseminación incrementa. En el algoritmo de *inundación* el porcentaje decae más rápido que en el *algoritmo propuesto* como consecuencia del mayor tráfico generado por la técnica de inundación y que hace que el rendimiento de la red disminuya. Respecto a los algoritmos de *Kim* y de *Butenko y Ursulenko* el *algoritmo propuesto* es más efectivo ya que tiene un mayor porcentaje de paquetes exitosamente entregados.

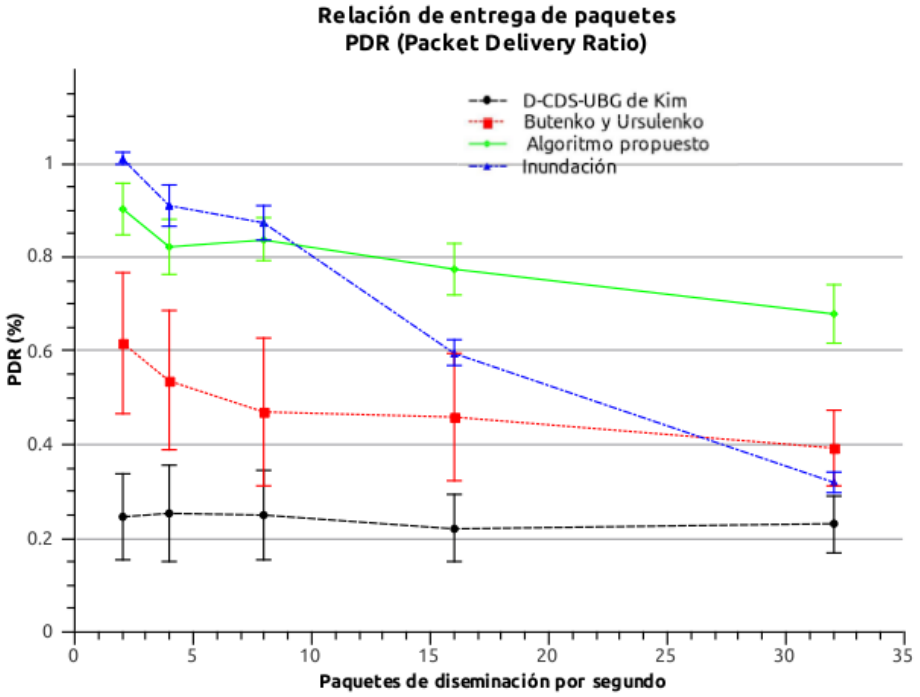

Figura 6.16: Sobrecarga de control en escenarios con variación en el número de paquetes de diseminación.

Retardo de extremo a extremo

Las gráficas de las Figuras 6.17, 6.18, 6.19, 6.20 y 6.21 nos muestran el retardo promedio que toma a un paquete llegar a su nodo destino a partir de su nodo origen. Cuando el tráfico de diseminación es bajo, el retardo promedio para el algoritmo de *inundación* y para el *algoritmo propuesto* es muy similar, incluso puede ser menor en la técnica de inundación debido a que un nodo recibe por primera vez un paquete de su camino menos costoso o del más corto. Conforme el tráfico de diseminación aumenta observamos que para el algoritmo de *inundación* el retardo incrementa más según el número de saltos que al paquete le toma llegar por primera vez a un nodo.

En las gráficas de las Figuras 6.20 y 6.21 en el algoritmo de *inundación* el retardo es de más de un segundo llegando hasta alrededor de cuarenta segundos, mientras que en el *algoritmo propuesto* el retardo no sobrepasa un segundo al igual que en los algoritmos de *Kim* y de *Butenko y Ursulenko*. Este resultado es debido a que en el algoritmo de *inundación* se esta generando mucho tráfico en la red y, por lo tanto, a un paquete le cuesta mucho llegar a un

destino dando un mayor número de saltos que en el *algoritmo propuesto* aunque el paquete siga el camino menos costoso.

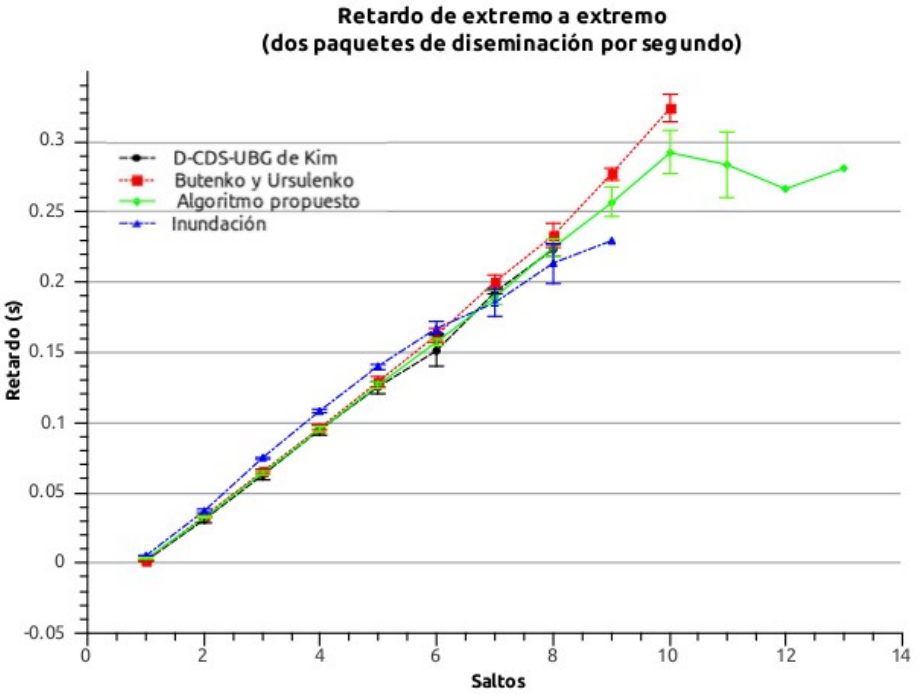

Figura 6.17: Retardo de extremo a extremo (dos paquetes por segundo).

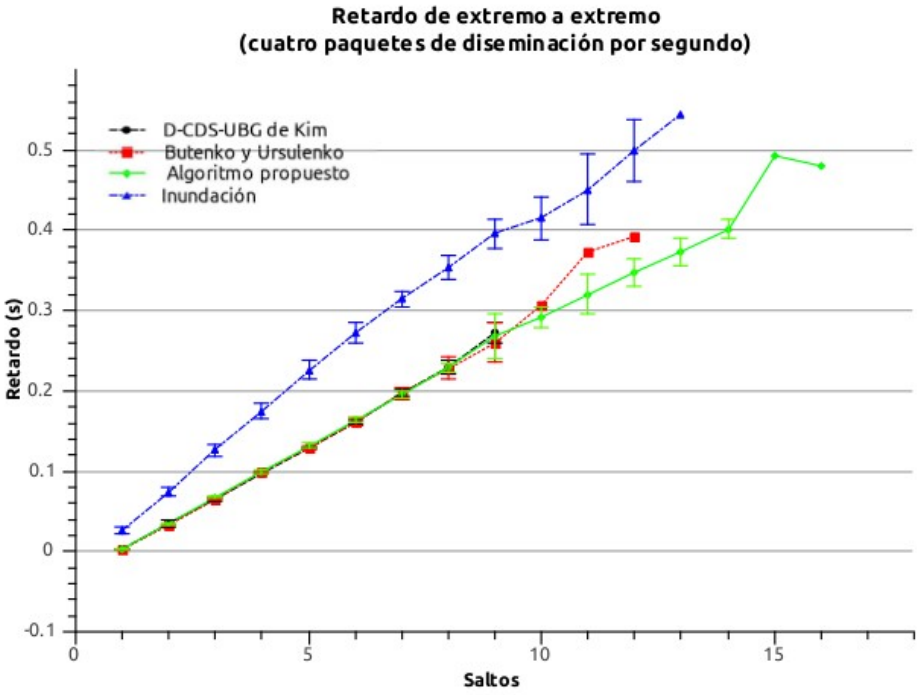

Figura 6.18: Retardo de extremo a extremo (cuatro paquetes por segundo).

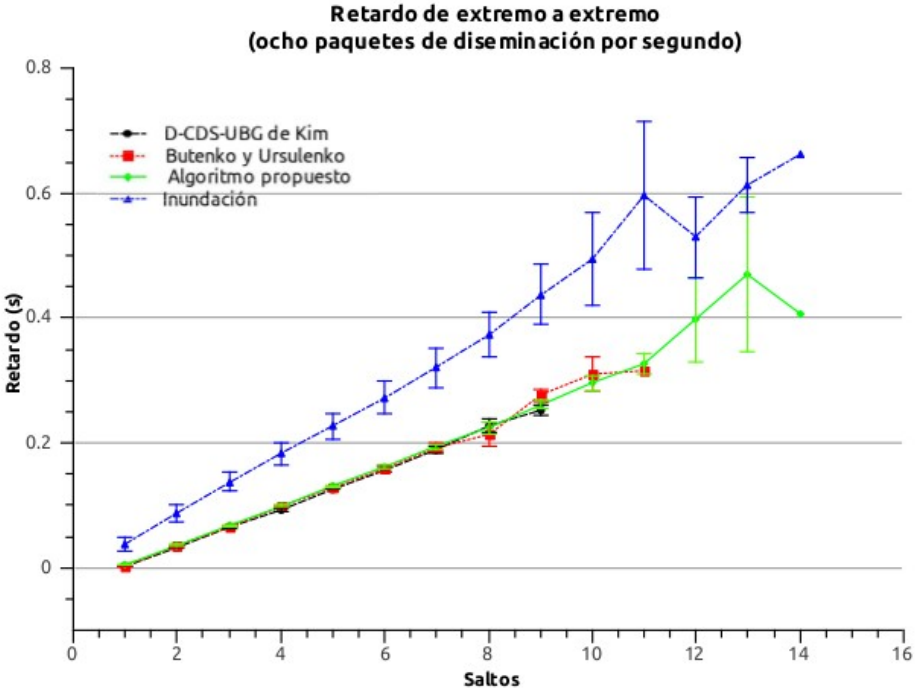

Figura 6.19: Retardo de extremo a extremo (ocho paquetes por segundo).

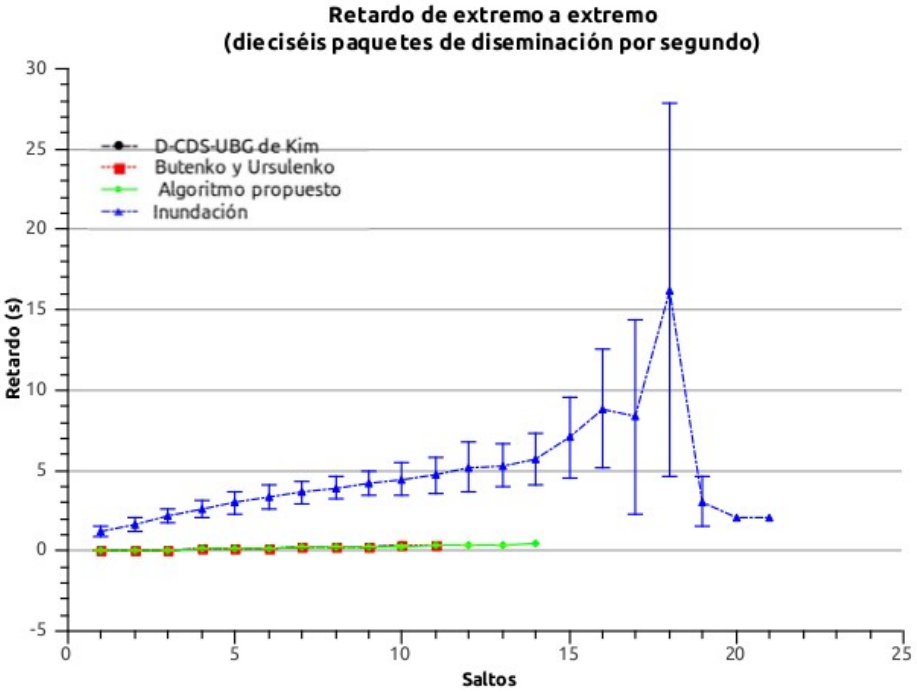

Figura 6.20: Retardo de extremo a extremo (dieciséis paquetes por segundo).

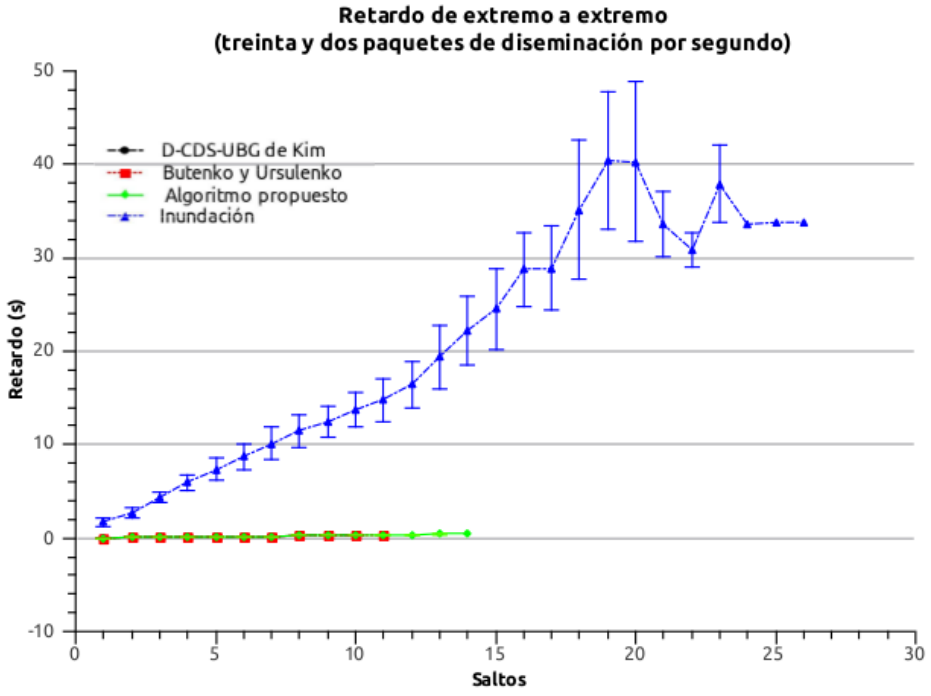

Figura 6.21: Retardo de extremo a extremo (treinta y dos paquetes por segundo).

Sobrecarga total

La sobrecarga total, mostrada en la gráfica de la Figura 6.22, aumenta para todos los algoritmos conforme se incrementa el número de paquetes de diseminación. Sin embargo, el número de paquetes utilizados en el algoritmo de *inundación* sigue siendo mayor aunque el porcentaje de paquetes exitosamente recibidos disminuya como se muestra en la gráfica de la Figura 6.16.

Una vez más este resultado refleja el alto tráfico de paquetes en la red que se originan al utilizar la técnica de inundación y que trae como consecuencia un bajo desempeño de la red al hacer uso de muchos recursos como lo es el ancho de banda y el espacio en colas.

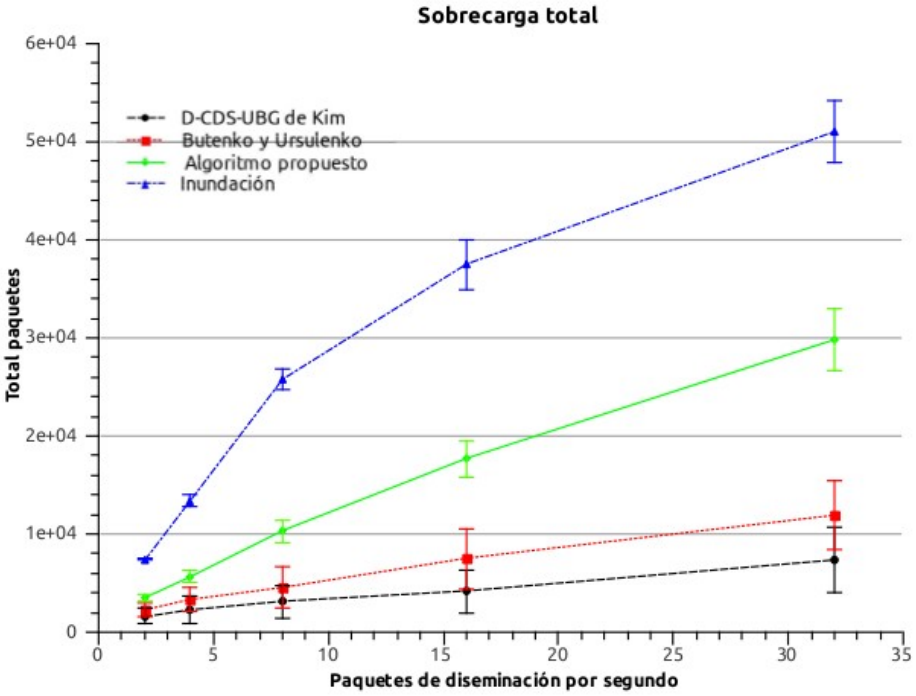

Figura 6.22: Sobrecarga total en escenarios con variación en el número de paquetes de diseminación.

- **Sobrecarga de datos**

  La gráfica de la Figura 6.23 nos muestra la diferencia en el número de paquetes de diseminación transmitidos en la técnica de inundación y los transmitidos por el *algoritmo propuesto*, el algoritmo de *Kim* y el de *Butenko y Ursulenko*. En este caso el *algoritmo propuesto* resulta más eficiente al tener una menor sobrecarga de datos con

un mayor porcentaje de paquetes exitosamente recibidos como se muestra en la gráfica de la Figura 6.16.

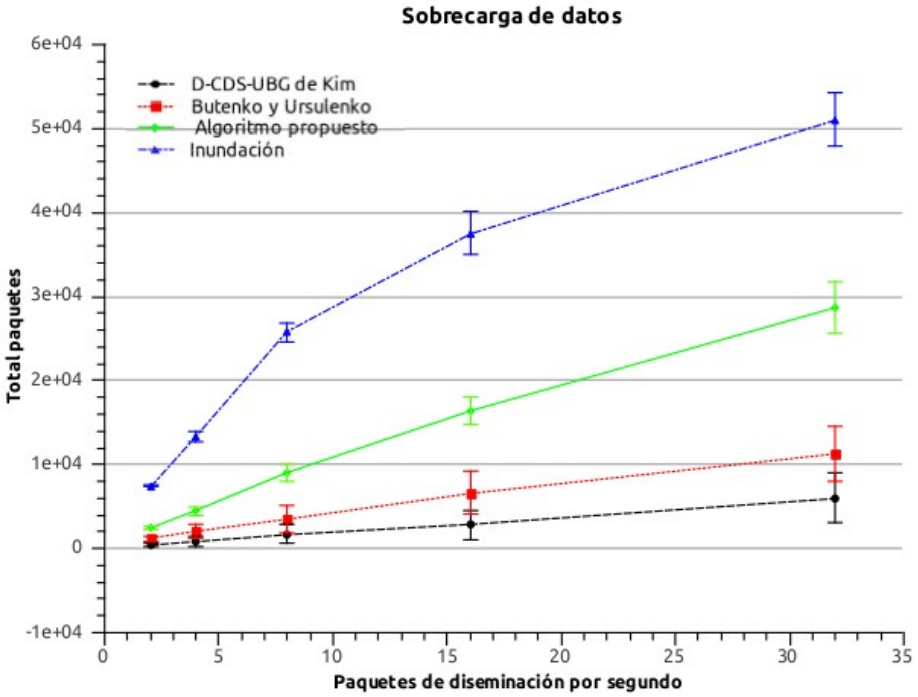

Figura 6.23: Sobrecarga de datos en escenarios con variación en el número de paquetes de diseminación.

- **Sobrecarga de control**

  La gráfica de la Figura 6.24 nos muestra la diferencia en el número de paquetes de control transmitidos por los algoritmos de *Kim*, de *Butenko y Ursulenko* y por el *algoritmo propuesto*. La gráfica muestra que el *algoritmo propuesto* tiene menor sobrecarga que el algoritmo de *Kim* aún cuando es más efectivo respecto a la métrica de relación de entrega de paquetes.

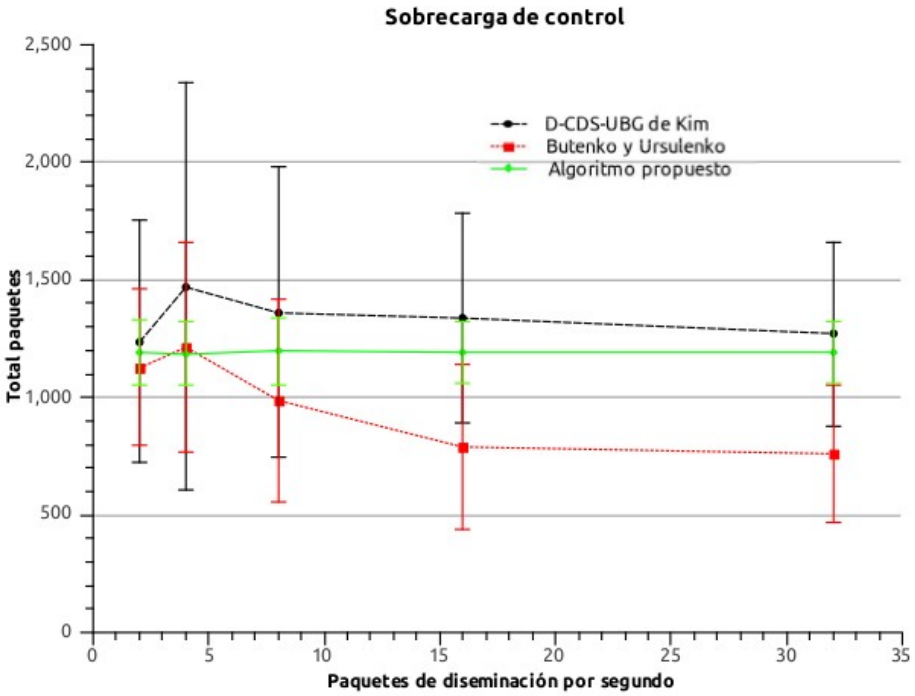

Figura 6.24: Sobrecarga de control en escenarios con variación en el número de paquetes de diseminación.

Capítulo 7

Conclusiones

En este último capítulo se presentan las conclusiones deducidas a partir de la investigación y del trabajo desarrollados en esta tesis así como sus principales aportaciones y los trabajos a futuro.

Conclusiones

Se logró la implementación de un algoritmo que en general mejora el desempeño de los algoritmos existentes con el mismo objetivo. El *algoritmo propuesto* y los algoritmos con los que se comparó fueron implementados y simulados bajo las mismas condiciones en un ambiente más cercano a la realidad. El *algoritmo propuesto* se comportó mejor debido a que toma en cuenta muchos de los principales problemas que se presentan en una MANET, tal como la fallas en la capa MAC, movilidad de los nodos, etc.

Los resultados de las simulaciones que se presentan en las gráficas, muestran que el *algoritmo propuesto* tiene un mejor desempeño para los diferentes tipos de métricas. Por lo tanto, el *algoritmo propuesto* es más eficiente que los algoritmos de aproximación existentes, con el mismo propósito, bajo las métricas de desempeño para un algoritmo de diseminación de información en una MANET.

Una red ad hoc es escalable bajo el *algoritmo propuesto*. Ya que, en los resultados de las simulaciones para el escenario estático, el desempeño de la red se mantuvo casi constante desde los 25 hasta los 100 nodos que fueron simulados tal como se muestra en la gráfica de la Figura 6.1, donde el porcentaje de paquetes entregado se mantiene alrededor de un 95%. Esta métrica es de gran importancia ya que también nos da una idea de la calidad del conjunto

dominante. Es decir, entre mayor sea la cantidad de paquetes entregados, significa que el conjunto dominante construido cubre la mayor cantidad de nodos. Por lo tanto, el *algoritmo propuesto* construye conjuntos dominantes que cubren un mayor número de nodos en comparación a los algoritmos de *Kim* y de *Butenko y Ursulenko*.

Cuando existe un bajo tráfico de paquetes de diseminación los resultados para las métricas retardo de extremo a extremo y relación de entrega de paquetes favorecen la técnica de inundación pero cuando el tráfico aumenta el desempeño de la técnica de inundación disminuye, esto lo podemos observar en las gráficas de las Figuras 6.17, 6.18, 6.19, 6.20 y 6.21 para la métrica retardo de extremo a extremo y en 6.16 para la métrica relación de entrega de paquetes.

También observamos la baja eficiencia del algoritmo de *inundación* en las gráficas de las Figuras 6.6, 6.14 y 6.23, donde se muestra que esta técnica realiza una cantidad mayor de retransmisiones de paquetes de diseminación. Este resultado refleja el alto tráfico de paquetes en la red que se originan al utilizar la técnica de inundación y que trae como consecuencia un bajo desempeño de la red al hacer uso de muchos recursos como lo es el ancho de banda y el espacio en colas.

Los resultados del algoritmo de *inundación* nos permitieron observar el costo de los problemas conocidos como el problema de *tormenta de transmisión* y que son: alta redundancia, alta contención y mayor probabilidad de colisiones. El *algoritmo propuesto* reduce, por ejemplo, la redundancia en el cubrimiento de un nodo, esto se deduce de la comparación de paquetes de datos utilizados por el algoritmo de *inundación* y la menor cantidad de paquetes de datos utilizados por el *algoritmo propuesto*.

7.2. Principales aportaciones

- En esta tesis se realizaron por primera vez análisis experimentales de algoritmos que calculan de manera distribuida conjuntos dominantes conexos basados en el modelo de esfera unitaria usando un modelo de propagación 3D.
- También por primera vez se implementó en un simulador de red realista como NS2 los algoritmos que componen el estado del arte en algoritmos distribuidos que calculan

conjuntos dominantes conexos basados en el modelo de esfera unitaria.

- Otro aporte, es el desarrollo del *algoritmo propuesto* que tiene un desempeño mejor, en cuanto a tasa de entrega de paquetes y retardo a extremo a extremo, que los algoritmos que conforman el estado del arte en algoritmos distribuidos que calculan conjuntos dominantes conexos basados en el modelo de esfera unitaria y al mismo tiempo, conserva un factor de aproximación igual al algoritmo de *Kim*, la cual es la mejor actual en el calculo de conjuntos dominantes conectados en gráficas de esferas unitarias.

7.3 Trabajos a futuro

Como posible trabajo a futuro que se puede llevar a cabo después del trabajo presentado en esta tesis, está el cálculo del valor del factor de aproximación ρ del *algoritmo propuesto* para otros problemas como el de minimizar el camino más largo dentro del conjunto dominante conectado.

Bibliografía

[1] K.M. Alzoubi, P.J. Wan, y O. Frieder. New distributed algorithm for connected dominating set in wireless ad hoc networks. En System Sciences, 2002. HICSS. Proceedings of the 35th Annual Hawaii International Conference on, p´gs. 3849– 3855. IEEE, 2002.

[2] S. Butenko, S. Kahruman-Anderoglu, y O. Ursulenko. On connected domination in unit ball graphs. Optimization Letters, 5(2):195–205, 2011.

[3] S. Butenko y O. Ursulenko. On minimum connected dominating set problem in unit-ball graphs. Preprint Submitted to Elervier Science, 2007.

[4] M. Cardei, X. Cheng, X. Cheng, y D.Z. Du. Connected domination in multihop ad hoc wireless networks. En Proc. Sixth Int'l Conf. Computer Science and Informatics (CS&I'02). Citeseer, 2002.

[5] X. Cheng, X. Huang, D. Li, W. Wu, y D.Z. Du. A polynomial-time approximation scheme for the minimum-connected dominating set in ad hoc wireless networks. Networks, 42(4):202 208, 2003.

[6] I. Cidon y O. Mokryn. Propagation and leader election in a multihop broadcast environment. Distributed Computing, págs. 104–118, 1998.

[7] B.N. Clark, C.J. Colbourn, y D.S. Johnson. Unit disk graphs. Discrete mathematics, 86(1):165–177, 1990.

[8] A. Das, M. Aasawat, C. Mandal, y C. Reade. An improved greedy construction of minimum connected dominating sets in wireless networks. En Wireless Communications and Networking Conference (WCNC), 2011 IEEE, págs. 790–795. IEEE, 2011.

[9] B. Das y V. Bharghavan. Routing in ad-hoc networks using minimum connected dominating sets. En Communications, 1997. ICC 97 Montreal,'Towards the Knowledge Millennium'. 1997 IEEE International Conference on, tomo 1, págs. 376–380. IEEE, 1997.

[10] R. Edwards y J. Durkin. Computer prediction of service area for VHF mobile radio networks. Proceedings of the IEE, 116(9):1493–1500, 1969.

[11] S. Filiposka y D. Trajanov. 3d simulations for wireless ad hoc networks in grid environment.

[12] M.R. Garey, R.L. Graham, y J.D. Ullman. Worst-case analysis of memory allocation algorithms. En Proceedings of the fourth annual ACM symposium on Theory of computing, págs 43–150. ACM, 1972.

[13] M.R. Garey y D.S. Johnson. Computers and intractability, tomo 174. Freeman San Francisco, CA, 1979.

[14] S. Guha y S. Khuller. Approximation algorithms for connected dominating sets. Algorithmica, 20(4):374–387, 1998.

[15] D.S. Hochba. Approximation algorithms for NP-hard problems. ACM SIGACT News, 28(2):40–52, 1997.

[16] R. Hoppe. Bemerkungen der redaktion zu'bender, c., bestimmung der gr o ten anzahl gleich grosser kugeln, welche sich auf eine kugel von demselben radius wie die ubrigen auflegen lassen. Arch. Math. Phys, 56:302–306, 1874.

[17] D.B. Johnson y D.A. Maltz. Dynamic source routing in ad hoc wireless networks. Mobile computing, pás. 153–181, 1996.

[18] D.S. Johnson. Approximation algorithms for combinatorial problems. Journal of computer and system sciences, 9(3):256–278, 1974.

[19] D. Kim, Z. Zhang, X. Li, W. Wang, W. Wu, y D.Z. Du. A better approximation algorithm for computing connected dominating sets in unit ball graphs. Mobile Computing, IEEE Transactions on, 9(8):1108–1118, 2010.

[20] Y. Li, M.T. Thai, F. Wang, C.W. Yi, P.J. Wan, y D.Z. Du. On greedy construction of connected dominating sets in wireless networks. Wireless Communications and Mobile Computing, 5(8):927–932, 2005.

[21] H. Lim y C. Kim. Flooding in wireless ad hoc networks. Computer Communications, 24(3):353–363, 2001.

[22] C. Lund y M. Yannakakis. On the hardness of approximating minimization problems. Journal of the ACM (JACM), 41(5):960–981, 1994.

[23] M. Min, H. Du, X. Jia, C.X. Huang, S.C.H. Huang, y W. Wu. Improving construction for connected dominating set with Steiner tree in wireless sensor networks. Journal of Global Optimization, 35(1):111–119, 2006.

[24] C.E. Perkins y E.M. Royer. Ad-hoc on-demand distance vector routing. En Mobile Computing Systems and Applications, 1999. Proceedings. WMCSA'99. Second IEEE Workshop on, págs. 90–100. IEEE, 1999.

[25] K.H. Rosen. Discrete mathematics and its applications. McGraw-Hill Boston, MA, 1999.

[26] L. Ruan, H. Du, X. Jia, W. Wu, Y. Li, y K.I. Ko. A greedy approximation for minimum connected dominating sets. Theoretical Computer Science, 329(1):325– 330, 2004.

[27] Y.C. Tseng, S.Y. Ni, Y.S. Chen, y J.P. Sheu. The broadcast storm problem in a mobile ad hoc network. Wireless networks, 8(2):153–167, 2002.

[28] P.J. Wan, K.M. Alzoubi, y O. Frieder. Distributed construction of connected dominating set in wireless ad hoc networks. En INFOCOM 2002. Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE, tomo 3, págs. 1597–1604. IEEE, 2002.

[29] D.P. Williamson y D.B. Shmoys. The design of approximation algorithms. Cambridge Univ Pr, 2011.

[30] J. Wu. Extended dominating-set-based routing in ad hoc wireless networks with unidirectional links. Parallel and Distributed Systems, IEEE Transactions on, 13(9):866–881, 2002.

[31] Z. Zhang, X. Gao, W. Wu, y D.Z. Du. Ptas for minimum connected dominating set in unit ball graph. Wireless Algorithms, Systems, and Applications, págs. 154–161, 2008.

[32] X. Zhong, J. Wang, y N. Hu. Connected dominating set in 3-dimensional space for ad hoc network. En Wireless Communications and Networking Conference, 2007. WCNC 2007. IEEE, págs. 3609–3612. IEEE, 2007.

[33] C. Zong y J. Talbot. Sphere packings. Springer Verlag, 1999.

Apéndice A

Glosario

A continuación se presenta la terminología usada a lo largo de esta tesis.

Sea una gráfica $G = (V, E)$, donde V es el conjunto de vértices y E el conjunto de aristas.

N(u) se llama *conjunto de vecinos abierto*, que es el conjunto de nodos vecinos de un nodo u y se define como $N(u) = \{v \mid (u, v) \text{ pertenece a } E\}$.

N[u] se llama *conjunto de vecinos cerrado* y se define como: $N[u]=N(u)\cup\{u\}$

N(C) es el conjunto de nodos vecinos para un conjunto de nodos C y se define como $N(C)=(\cup_{x\in C}N(x))\setminus C$.

C(u) se llama *conjunto de nodos hijos* de u, que son nodos vecinos marcados como hijos para el nodo u.

P(u) se llama *nodo padre* de u, que es un nodo vecino marcado como padre para el nodo u.

Δ se llama *grado máximo de una gráfica*, que es el grado máximo de un vértice en una gráfica, o bien el máximo número de aristas adyacentes a un nodo.

Conjunto independiente, es un subconjunto de V tal que no existen dos vértices dentro del subconjunto que sean adyacentes.

Conjunto independiente maximal o MIS (*Maximal Independent Set*), es un subconjunto independiente tal que, la adición de cualquier otro vértice fuera del conjunto rompe la propiedad de independencia del conjunto. De esta forma, cualquier vértice fuera del conjunto independiente maximal debe ser adyacente a un algún nodo en el conjunto.

$M_{v,C}$ es un MIS de nodos adyacentes a v y que no están en el conjunto C dado.

Conjunto dominante, es un subconjunto $V' \subseteq V$ tal que, para todo $u \in V - V'$ existe al

menos un nodo $v \in V'$ tal que $(u, v) \in E$. Los vértices en *V'* son conocidos como *dominadores* o *nodos dominantes* y los vértices en *V* - *V'* como *nodos dominados* o *dominatees*. Podemos observar que cada MIS es un conjunto dominante. Sin embargo, ya que los nodos en un conjunto dominante pueden ser adyacentes entre sí, no todo conjunto dominante es un MIS. Encontrar un conjunto dominante de tamaño mínimo o MDS (*Minimum Dominating Set*) es un problema NP-difícil [13].

Conjunto dominante conectado (CDC), es un conjunto dominante de *G* que forma un subgráfica conexa. Uno de los enfoques utilizados para la construcción de un CDC consiste en encontrar un MIS y añadir vértices adicionales, si es necesario, para conectar los nodos en el MIS.

Conjunto dominante conectado mínimo (CDCM), es el CDC de cardinalidad mínima. Encontrar un CDCM también es NP-difícil [13].

$\gamma(G)$ se llama *número de dominación* y se define como: el número de vértices en el conjunto dominante más pequeño.

El *problema del conjunto dominante* es un problema de decisión NP-completo [13] que prueba si existe $\gamma(G) \leq k$ para una entrada *k*. El problema de optimización relacionado consiste en encontrar el valor de $\gamma(G)$.

El *problema del árbol de Steiner* es un problema NP-difícil [13] que se define de la siguiente manera: dado un subconjunto de vértices *V'* en una gráfica de aristas ponderadas, encontrar un árbol de peso mínimo que abarque el subconjunto de vértices *V'*, el árbol puede incluir otros vértices aparte de los vértices en *V'*. El problema del árbol de Steiner con nodos ponderados es esencialmente el mismo problema, excepto que los vértices de la gráfica tienen pesos asociados a ellos y el peso del árbol es la suma de los pesos de sus vértices. El árbol de Steiner con nodos unitarios ponderados es el caso especial en que todos los vértices, fuera de *V'*, tienen el mismo peso. Todos los vértices en *V'* tienen peso igual a cero. Después de encontrar un MIS, conectar los nodos se puede formular como una instancia del problema del árbol de Steiner.

Gráfica de discos unitarios o UDG (*Unit Disk Graph*), es la gráfica de intersecciones de un conjunto de círculos unitarios en el plano euclidiano. Es decir, se forma un vértice de cada círculo, y existe una arista entre dos vértices cuando los círculos correspondientes se cruzan entre sí.

Gráfica de esferas unitarias o UBG (*Unit Ball Graph*), es la gráfica de intersecciones de un conjunto de esferas unitarias en el espacio euclidiano. Es decir, se forma un vértice de cada esfera, y existe una arista entre dos vértices cuando las esferas correspondientes se cruzan entre sí.