UNIVERSIDAD NACIONAL AUTÓNOMA
DE MÉXICO

FACULTAD DE CIENCIAS

ALGORITMOS SOBRE GRAFICAS PARA COMPACTACION DE AUTOMATAS FINITOS

T E S I S

QUE PARA OBTENER EL TÍTULO DE:

LICENCIADO EN CIENCIAS DE LA COMPUTACION

P R E S E N T A :

DIEGO RABAGO ARREDONDO

DIRECTORA:
DRA. ELISA VISO GUROVICH

2008

VNIVERSIDAD NACIONAL AVTÓNOMA DE MEXICO

FACULTAD DE CIENCIAS
Secretaría General
División de Estudios Profesionales

Votos Aprobatorios

Act. Mauricio Aguilar González
Jefe de la División de Estudios Profesionales
Facultad de Ciencias
P r e s e n t e

Por este medio hacemos de su conocimiento que hemos revisado el trabajo escrito titulado:

Algoritmos sobre gráficas para compactación de autómatas finitos

realizado por **Rábago Arredondo Diego** con número de cuenta **4-0104750-1** quien ha decidido titularse mediante la opción de **tesis** en la licenciatura en **Ciencias de la Computación**., Dicho trabajo cuenta con nuestro voto aprobatorio.

Propietario Tutora	Dra. Elisa Viso Gurovich	
Propietario	Dr. Sergio Rajsbaum Gorodezky	
Propietario	Dr. Jorge Urrutia Galicia	
Suplente	Dr. José de Jesús Galaviz Casas	
Suplente	M. en I. María de Luz Gasca Soto	

Atentamente,
"POR MI RAZA HABLARÁ EL ESPÍRITU."
Ciudad Universitaria, D. F., a 18 de abril de 2008
EL COORDINADOR DEL COMITÉ ACADÉMICO DE LA LICENCIATURA EN CIENCIAS DE LA COMPUTACIÓN

DR. JOSÉ DE JESÚS GALAVIZ CASAS

FACULTAD DE CIENCIAS
CONSEJO DEPARTAMENTAL

Señor sinodal: antes de firmar este documento, solicite al estudiante que le muestre la versión digital de su trabajo y verifique que la misma incluya todas las observaciones y correcciones que usted hizo sobre el impreso.

Índice General

Índice de Figuras

Índice de Tablas

Capítulo 1

Introducción

El apareamiento de cadenas, un problema clásico en las ciencias de la computación, ha sido estudiado ampliamente durante las últimas décadas, entregando un sinnúmero de algoritmos que solucionan, con diferentes aproximaciones, distintas versiones del problema. En la versión más simple, se busca encontrar todas las presencias de una cadena determinada sobre un texto fijo, siendo las demás versiones variaciones de este problema original. En algunas de ellas se busca aparear una cadena sobre un texto dinámico, en otras se utiliza un conjunto de varias cadenas y se intenta encontrar todas las presencias de ellas sobre el campo de búsqueda, otras más encuentran presencias aproximadas de un cierto grupo de cadenas. Además de los casos mencionados, existen diversas variaciones del problema original, algunas de las cuales siguen siendo estudiadas ya sea para encontrar nuevas maneras de atacar el problema o mejorar, a partir de diferentes técnicas, las aproximaciones anteriores. De la misma forma que las soluciones, se han encontrado también distintas aplicaciones para dichos algoritmos, desde búsquedas bibliográficas hasta el análisis de secuencias en computación molecular.

Una de estas aplicaciones se enfoca a las redes de computadoras, campo en el que se utiliza el apareamiento de cadenas para mejorar la seguridad de las redes y facilitar la distribución de información dentro de éstas. Esto se logra a través de la inspección profunda de los datos contenidos en los paquetes que conforman los flujos de entrada y comparando su contenido contra un conjunto finito de cadenas, denotadas habitualmente como patrones.

El desarrollo de nuevas tecnologías ha permitido, a su vez, la generación de dispositivos de red cada vez más eficientes, capaces de examinar los flujos de datos a velocidades crecientes; el reto reside en mantener la velocidad en que se inspeccionan los datos cercana a las velocidades de transmisión más elevadas en la industria, para no convertirse en un cuello de botella en el sistema. Para la integración de estos dispositivos a los sistemas de redes computacionales, se imponen ciertas limitantes en cuanto a la memoria disponible para ello y la capacidad de carga que pueden analizar. Dadas estas limitantes, se ha buscado desarrollar nuevos algoritmos que cumplan con los requerimientos solicitados, disminuyendo las necesidades de memoria para almacenar las estructuras y datos que se utilicen, así como para amplificar la cantidad de datos transmitida por el canal.

Para lograrlo, en años recientes se ha generado un alto interés en migrar de esquema, intercambiando los conjuntos de cadenas por conjuntos de expresiones regulares, dada la alta expresividad y flexibilidad que ofrecen estas últimas. Las complicaciones emanan esencialmente de los altos requerimientos de memoria que se necesitan para implementar estas expresiones. Un método ampliamente utilizado para representar expresiones regulares son los autómatas finitos deterministas (AFD), pues cuentan con la funcionalidad agregada de que se puede construir un autómata con el mínimo número de estados para cualquier expresión regular que se presente. Lo mismo puede lograrse para un autómata que represente, no sólo a una, sino a un conjunto de expresiones regulares. No obstante, cuando se utiliza este enfoque para analizar datos en redes computacionales los autómatas resultantes son demasiado grandes para ser alojados en dispositivos con limitantes de memoria. Esto se debe en gran parte al tamaño de los alfabetos utilizados, que generalmente contienen un número mayor o igual a 256 carácteres, resultando en el mismo número de transiciones salientes de cada estado en el autómata, cada una de las cuales debe ser codificada para representar el AFD correctamente. Dada la migración de muchos proveedores de equipo a este esquema, últimamente se han buscado desarrollar enfoques que permitan mejorar el desempeño de los dispositivos, tanto disminuyendo las necesidades de memoria como aumentando la capacidad de carga.

En su artículo "Algorithms to Accelerate Multiple Regular Expressions Matching for Deep Packet Inspection", Kumar et al. (2006) [11] presentan una solución a este problema al implementar el apareamiento basado en expresiones regulares sobre un canal de entrada de datos. Se describe ahí una variación a los autómatas finitos deterministas que permite reducir drásticamente los requisitos en términos de memoria para representar el conjunto de expresiones regulares. Esencialmente, se compacta el AFD al eliminar un gran número de sus transiciones buscando eliminar la redundancia de información. Los autores presentan dos algoritmos con diferentes características para desarrollar esta tarea, uno de los cuales es un refinamiento del otro. En este trabajo revisaremos las estructuras y algoritmos planteados por Kumar et al [11].

Adicionalmente, presentamos dos nuevos algoritmos sobre gráficas para compactación de autómatas finitos. Cada uno de estos sigue un método diferente al intentar obtener estructuras con el mejor desempeño posible en cuanto a requerimientos de memoria y tiempos de ejecución. Se enuncian y analizan las principales propiedades de cada algoritmo. Simultáneamente, se realiza un estudio comparativo entre el desempeño de los cuatro algoritmos al ejecutarse sobre gráficas con diferentes características. Uno de los nuevos algoritmos ofrece importantes mejoras de desempeño en comparación al de los algoritmos propuestos por los autores en autómatas con características similares a los utilizados en las aplicaciones a redes informáticas.

Capítulo 2

Conceptos preliminares

En este capítulo se presentan brevemente algunos conceptos fundamentales necesarios para comprender en lo sucesivo el resto de este trabajo. El enfoque que se da a estos conceptos preliminares es sucinto y no se dedica demasiado espacio en explicarlos, sino que simplemente se enlistan las definiciones y propiedades requeridas. Se estudian primero algunos conceptos elementales de teoría de gráficas para continuar con una breve introducción a los autómatas finitos y a las expresiones regulares.

2.1 Conceptos básicos de la teoría de gráficas

2.1.1 Gráficas

Una gráfica G es una pareja (V, E) de conjuntos que satisfacen $\forall e \in E, e = (u, v)$ en donde $u, v \in V$ – es decir que cada elemento de E es a su vez un subconjunto de 2 elementos de V – y V es no vacío. Llamamos vértices o nodos a los elementos de V y aristas a los elementos de E. En una gráfica una arista de un vértice v a un vértice u es lo mismo que una arista que va de u a v, es decir, en la definición de una arista es indistinto el orden en que se indiquen los vértices. Habitualmente, una gráfica se relaciona con su representación gráfica, en la cual los vértices se representan como puntos y las aristas como arcos que unen los vértices correspondientes. Se dice que dos vértices u y v son adyacentes si existe una

arista $e = (u, v) \in E$ y se dice que entonces la arista e es incidente en un vértice v si v es uno de los vértices de e. El orden de una gráfica se refiere al número de vértices que contiene y se denota por $\mid G \mid$ o bien $\mid V \mid$. El grado o la valencia de un vértice v se define como el número de aristas que inciden en v, y se denota por $\delta(v)$. Decimos que un vértice es aislado si cumple que su valencia es 0. El grado mínimo en una gráfica G se denota como $\delta(G)$ y el grado máximo como $\Delta(G)$; Se definen de la siguiente manera:

$$\delta(G) = min\{\delta(v) \mid v \in V(G)\}$$
$$\Delta(G) = max\{\delta(v) \mid v \in V(G)\}$$

Podemos también obtener el grado promedio de una gráfica sumando cada grado individual y dividiendo entre el número total de vértices.

2.1.2 Caminos, trayectorias y ciclos

Un camino es una sucesión de aristas y sus respectivos vértices de la forma $(v_0,\ v_1)$, $(v_1,\ v_2)$, ... , $(v_{n-1},\ v_n)$ en la cual se permite la repetición tanto de aristas como de vértices. Una trayectoria o camino simple es un camino en el que no se repiten vértices, o formalmente, una gráfica $C = (V, E)$ en la cual $V = \{v_0, v_1, ..., v_n\}$, $E = \{(v_0, v_1), (v_1, v_2), ..., (v_{n-1}, v_n)\}$ y cada $v_i \neq v_j$ si $i \neq j$. Decimos que v_0 es el primer vértice, v_n el último vértice de la trayectoria y v_1 hasta v_{n-1} son vértices interiores. Podemos denotar a las trayectorias con una letra mayúscula (generalmente P) e identificarlas con sus vértices, esto es, $P = v_0v_1...v_n$. Una trayectoria que va de un punto X a un punto Y se denomina como una XY-trayectoria, mientras que nos referimos a la longitud de una trayectoria como el número de aristas que contiene. Para cada vértice v, se denota por $d(v)$ a la longitud de la trayectoria más larga desde v a cualquier otro vértice de la gráfica. Se dice que un gráfica es conexa si existe una trayectoria entre cualquier par de vértices. El diámetro de una gráfica se refiere a la longitud de cualquier trayectoria maximal dentro de ella, o lo que es lo mismo, la mayor distancia entre cualesquiera dos vértices de la gráfica. Se utilizará la notación $diam(G)$ para denotar al diámetro de una gráfica G cuando esto sea necesario. Finalmente, un ciclo o circuito es un

camino simple y cerrado que se forma al agregar una arista que incida en el primer y último vértices de una trayectoria.

2.1.3 Digráficas y multigráficas

Una gráfica dirigida o digráfica es una gráfica $G = (V, E)$ en la cual los elementos de E son pares ordenados o lo que es lo mismo, las aristas son dirigidas. Se dice que una arista $e = (u, v)$ va del vértice u al vértice v. Un camino sobre una digráfica considera la dirección de las aristas que lo conforman. Una multigráfica es una gráfica en la que se permiten múltiples aristas entre un mismo par de vértices y lazos (aristas que van al mismo vértice del que salen).

2.1.4 Árboles y bosques

Un árbol es una gráfica conexa y acíclica (es decir que no contiene ciclos). Un bosque es un conjunto de n árboles con $n \geq 2$. Algunas propiedades de los árboles se enlistan a continuación, si $T = (V, E)$ es un árbol, entonces se cumplen:

- Si $\mid V \mid = n$, entonces $\mid E \mid = n - 1$.
- Si $u, v \in V$ entonces existe una única uv-trayectoria.
- Agregar una arista al conjunto E crea un ciclo en la gráfica.
- Si se quita cualquier arista, T deja de ser conexa.

Un árbol dirigido es una digráfica que permanece acíclica y conexa si se ignora la orientación de sus aristas. Un árbol con raíz se forma cuando se distingue a un vértice de un árbol dirigido como la raíz del árbol; la orientación de las aristas suele alejarse de la raíz. Bajo esta convención u es el padre de v si $\exists (u, v) \in E$, y a su vez, la longitud de la trayectoria de u a la raíz es menor a la longitud de la trayectoria de v a la raíz. De la misma forma v es entonces hijo de u. Los vértices sin hijos se llaman hojas.

2.2 Autómatas finitos

Un autómata es un modelo matemático de una máquina de estados. Dichas máquinas funcionan al recibir como entrada una cadena de símbolos de un conjunto finito conocido como alfabeto de entrada, la cual las hace cambiar de estado en conformidad con una función de transición que define el comportamiento de la máquina. Estos autómatas tienen un estado distinguido conocido como estado inicial, en el cual se encuentra antes de empezar a consumir la entrada. Tienen también un conjunto de estados finales que determinan si la cadena leída es un elemento del lenguaje que la máquina reconoce. Un autómata finito modela una máquina con un número finito de estados.

Formalmente, un autómata finito se define como una quíntupla $(Q, \Sigma, \delta, q_0, F)$ donde:

- Q es un conjunto finito de estados.
- Σ es el alfabeto de entrada, un conjunto finito de símbolos.
- δ es la función de transición.
- q_0 es el estado inicial.
- F es el conjunto de estados finales con $F \subseteq Q$.

Una forma natural de representar a un autómata, y particularmente su función de transición, es a partir de un diagrama de transiciones. En estos diagramas el autómata se representa como una digráfica con aristas etiquetadas, en la cual los vértices representan los estados de Q mientras que las aristas dirigidas están definidas por la función de transición y etiquetadas con el símbolo de Σ correspondiente a cada transición. Se distingue al estado inicial con un arco entrante que no proviene de ningún otro estado y a los estados finales por estar representados con un doble círculo. Así, el autómata queda definido completamente.

2.2.1 Autómatas finitos deterministas

Un autómata finito determinista, o AFD, es un autómata finito cuya función de transición define una única transición para cada pareja (q, a) en donde $q \in Q, a \in \Sigma$. Es decir que δ

está definida como:

$$\delta : Q \times \Sigma \to Q$$

2.2.2 Autómatas finitos no deterministas

A diferencia de los AFD, en los autómatas finitos no deterministas, o AFN, cada estado puede tener más de una transición para un conjunto de símbolos del alfabeto de entrada. Inclusive pueden no tener transiciones de salida para uno o más símbolos Σ. Es decir que en este caso la función de transición es:

$$\delta : Q \times \Sigma \cup \{\varepsilon\} \to \wp(Q)$$

Vemos que en las transiciones es válido aceptar la cadena vacía, esto nos dice que no necesariamente debemos leer un símbolo para activar un nuevo estado o un conjunto de ellos y se conoce a éstas como transiciones-ε. Además, se dice que un AFN acepta una cadena de símbolos si existe alguna trayectoria desde q_0 a algún estado final leyendo dicha cadena.

2.2.3 Equivalencia entre AFD y AFN

Podría pensarse que los AFN tienen un poder expresivo superior al de los AFD o bien que pueden aceptar lenguajes que los autómatas deterministas no pueden. En esta sección se muestra que esto no es así.

Se dice que dos autómatas son equivalentes si aceptan el mismo lenguaje, esto es, el mismo conjunto de cadenas. La equivalencia entre un AFD y un AFN que aceptan lenguajes finitos es fácil de comprobar pues se pueden comparar las cadenas que aceptan ambos; pero cuando el lenguaje aceptado es infinito este "método" se vuelve inviable. Afortunadamente se tiene una demostración de dicha equivalencia con la propiedad de ser constructiva. En otras palabras no sólo afirma que existen autómatas deterministas y no deterministas equivalentes, sino que da las reglas para construirlos. Sin entrar en detalles podemos ver que todo AFD es también un AFN por lo que dado cualquier AFD podemos construir un AFN equivalente:

él mismo. Ahora bien, dado un AFN $M = \{Q, \Sigma, \delta, q_0, F\}$ podemos construir un el AFD asociado $\bar{M} = \{\bar{Q}, \bar{\Sigma}, \bar{\delta}, \bar{q_0}, \bar{F}\}$ dado por:

- $\bar{Q} = \wp(Q)$
- $\bar{\Sigma} = \Sigma$
- $\bar{\delta}(A, a) = \cup_{q \in A} \delta(q, a)$
- $\bar{q_0} = \{q_0\}$
- $\bar{F} = \{A \in \wp(Q) \mid A \cap F \neq \emptyset\}$

De esta demostración, resulta directo que los autómatas finitos deterministas y los no deterministas pueden aceptar los mismos lenguajes. Es decir, no existe un lenguaje aceptado por un AFN que no sea aceptado por un AFD y viceversa.

2.3 Expresiones regulares

Si se consideran conjuntos de cadenas que se pueden formar con un alfabeto Σ y algunas operaciones sobre estos conjuntos, obtenemos lo que se conoce como conjuntos o expresiones regulares (ER). La noción constructiva de estas expresiones comienza especificando las constantes (conjuntos básicos) y los operadores, a partir de los cuales se estructura un álgebra de Kleene. Éstos se describen a continuación:

- $\emptyset$, el conjunto vacío, es una ER.
- ε, la cadena o palabra vacía denotando al conjunto $\{\varepsilon\}$, es una ER.
- $a \in \Sigma$, un símbolo o caracter del alfabeto denotando al conjunto $\{a\}$, es una ER.

Mientras que, si R_1 y R_2 son expresiones regulares, los operadores son:

- $R_1 R_2$ o $R_1 \cdot R_2$ es una ER (concatenación).
- $R_1 \mid R_2$ o $R_1 + R_2$ es una ER (unión).

- R_1* es una ER (cerradura de Kleene).

Se mencionó arriba que las ER son un álgebra de Kleene sobre Σ por lo cual satisfacen los axiomas de este tipo de álgebras, es decir, suponiendo que R, S y T sean ER de un alfabeto Σ, tenemos que cumplen lo siguiente:

- Asociatividad de $\mid$ y $\cdot$, esto es, $R \mid (S \mid T) = (R \mid S) \mid T$ y $R(ST) = (RS)T$
- Conmutatividad de $\mid$, esto es, $R \mid S = S \mid R$
- Distributividad: $R(S \mid T) = RS \mid RT$, y de la misma forma $(R \mid S)T = RT \mid ST$
- ε como identidad de $\cdot$, es decir $\forall a \in \Sigma, a\varepsilon = \varepsilon a = a$
- $\emptyset$ como identidad de $\mid$, es decir que para cualquier ER R sobre Σ: $R \mid \emptyset = \emptyset \mid R = R$

Al evaluar las ER se considera que la mayor presedencia la tiene la cerradura de Kleene, seguida de la concatenación y finalmente la unión.

2.3.1 Lenguaje representado por las ER

Un lenguaje se define como un conjunto de cadenas sobre un cierto alfabeto. De esta definición es claro que cada ER representa un lenguaje pues puede construirse un conjunto de cadenas sobre el alfabeto a partir de ellas, y si la ER se denota por R el lenguaje que genera se denotará por L(R).

Ejemplo: Sea $\Sigma = \{a, b, c\}$ y sea $R = (a \mid b)(\varepsilon \mid c)*$ una ER sobre este alfabeto. Entonces $L(R) = \{a, b, ac, bc, acc, bcc, accc, bccc, \ldots\}$.

Las expresiones regulares pueden entonces expresar un tipo de lenguajes clasificados como *lenguajes regulares*, los cuales, a su vez, son los lenguajes que son aceptados por los autómatas finitos. Es decir que para cualquier lenguaje regular existe una ER que lo representa y un autómata finito que lo acepta, y recíprocamente, cada autómata finito acepta un lenguaje

regular y cada ER representa un lenguaje del mismo tipo. Además las ER corresponden en la jerarquía de Chomsky de gramáticas formales a gramáticas de tipo 3. Aunque una discusión explícita de estos conceptos sobrepasa el enfoque de este trabajo, nos interesa ampliamente el mapeo entre las ER y los AFN, que a su vez, por la equivalencia entre AFN y AFD nos muestra cómo representar las ER por medio de AFD.

2.3.2 Equivalencia entre AFD y expresiones regulares

Se mencionó arriba que los lenguajes representados por las ER y los aceptados por los autómatas finitos son los mismos; esto incita a pensar que existe una equivalencia entre autómatas finitos y ER. La demostración de ello es de hecho constructiva, pues dada cualquier ER se puede construir mediante un algoritmo bien definido su AFN asociado (y a partir de este último su AFD correspondiente) que acepte el lenguaje representado por la ER. Recíprocamente se puede también construir, a partir de cualquier AFD (y por lo mismo de cualquier AFN), una ER asociada que genere el mismo lenguaje que el autómata acepta. De estas dos construcciones, el interés práctico se centra en la primera pues los lenguajes se suelen representar intuitivamente por medio de ER y después se requieren autómatas que los puedan reconocer. Existen varios algoritmos para construir un AFN a partir de una ER; entre los más importantes se cuentan el de Thompson y el de Glushkov [13, 4], el primero por su simplicidad y el segundo por su eficiencia práctica. Esta eficiencia proviene del hecho de que el AFN generado por el algoritmo de Glushkov no genera transiciones-ε además de que el número de estados es lineal respecto a la cantidad de símbolos en la ER. Navarro y Raffinot [13] presentan una explicación detallada de ambos algoritmos.

Capítulo 3

Apareamiento de patrones orientado a redes de computadoras

3.1 Antecedentes

Dada una cadena P denotada como patrón y otra cadena T llamada texto, el problema esencial del apareamiento de patrones reside en encontrar todas las presencias del patrón P en el texto T. El amplio interés que se le ha dedicado a dicho problema durante las últimas décadas no sólo lo ha convertido en un problema clásico de las ciencias de la computación sino que también ha generado múltiples variaciones del problema y diferentes soluciones para todas ellas.

La variante más común del problema se refiere a encontrar las presencias en el texto de un conjunto de patrones $\{P_1, P_2, \ldots, P_n\}$ en lugar de cazar simplemente una única cadena [13, 8, 1, 18, 17]. También se han estudiado los casos en los que tanto los patrones como el texto puedan ser flujos de datos dinámicos, es decir, que puedan modificarse a medida que se procesan, generando así nuevas presencias de los patrones o bien eliminando algunas de las previamente reportadas[2, 14]. Recientemente, la utilización de textos de mucho mayor tamaño, así como las cantidades exorbitantes de patrones utilizadas en aplicaciones como el análisis de secuencias genéticas en biología computacional o el análisis del tráfico en redes

de computadoras, han advertido la necesidad de cambiar de esquema. El problema original se ha enfocado últimamente en encontrar métodos para reducir la cantidad de memoria necesaria para procesar los textos en busca de patrones, dando como resultado la creciente migración del tipo de patrones utilizados en las búsquedas, de cadenas comunes a cadenas extendidas y expresiones regulares [11, 19, 13]. Esfuerzos paralelos se han enfocado en los últimos tiempos en desarrollar soluciones para el problema de apareamiento aproximado de patrones, esto es, aceptando un cierto grado de diferencia entre los patrones buscados y los resultados que se entregan, siendo la motivación de esta aproximación la protección contra errores de transcripción de datos o de transmisión de información.

3.2 Orientación a las redes de computadoras

La inspección profunda de paquetes (o procesamiento de contenido) sobre un canal de transmisión de datos en las redes de computadoras ha demostrado su utilidad por su habilidad para identificar y clasificar el tráfico basado en su contenido. Muchos servicios críticos de las redes deben ser capaces de analizar el área de datos (*payload*) y no únicamente la información contenida en el ncabezado de cada paquete, ya sea para manejar o distribuir el tráfico eficazmente o bien por motivos de seguridad como, por ejemplo, la detección de intrusiones[11, 6]. El basto crecimiento de las redes informáticas y las necesidades de los usuarios modernos han causado que los métodos tradicionales de inspección superficial de paquetes (analizando simplemente el encabezado) hayan sido desplazados y los proveedores de servicios se encuentren hoy ante el problema de desarrollar e implementar nuevos métodos que puedan inspeccionar el área de datos de cada paquete y además lo hagan acorde a las velocidades actuales de transmisión. El desarrollo de dispositivos de red capaces de analizar el contenido de los paquetes es ya una realidad. Líderes en la industria como Cisco y 3Com han comenzado a estudiar cómo desarrollar circuitos integrados capaces de realizar esta tarea, haciendo grandes inversiones en este sentido.

Generalmente, la inspección profunda de paquetes requiere que cada byte tanto del encabezado como del área de datos de los paquetes sea comparado a un conjunto de patrones

determinados previamente. Los investigadores en la materia se han dado a la tarea de generar algoritmos eficientes que puedan inspeccionar los datos a alta velocidad compárandolos a conjuntos de cadenas conocidas que permitan realizar tareas específicas. Las primeras aproximaciones se centraron en extender los algoritmos clásicos para apareamiento exacto de un conjunto de cadenas y sus variaciones para adaptarlos a trabajar con flujos de datos en redes. Entre estos algoritmos se cuentan los de Aho-Corasick[1], Set-Horspool[13], Wu-Manber[18], Commentz-Walter[13], SBDM y SBOM[13], que a su vez son extensiones de algoritmos previos que solucionan con distintas características el problema elemental para una única cadena. Utilizando técnicas de compresión de datos y diferentes estructuras de datos preprocesadas se ha logrado alcanzar un alto rendimiento en la inspección de paquetes.

Enfoques más recientes han comenzado a proponer diferentes arquitecturas de hardware que sean capaces de realizar la tarea de inspeccionar los paquetes a velocidades mayores que las implementaciones en software que corren sobre procesadores convencionales. La idea es construir circuitos integrados para aplicaciones específicas (ASIC por sus siglas en inglés) con capacidades suficientes de procesamiento y memoria para realizar la inspección de paquetes por su cuenta, es decir, dispositivos independientes que puedan cargar con todo el trabajo o con la mayor parte de éste. Junto con las propuestas de estas arquitecturas también se han desarrollado algoritmos para explotarlas al máximo, cada uno con características diferentes que esencialmente buscan reducir la complejidad ya sea en términos de memoria, de tiempo de ejecución o ambos.

El cambio de enfoque mencionado arriba al problema de inspeccionar el tráfico en las redes de computadoras no es el único. Uno más se basa en que el crecimiento de los conjuntos de reglas o patrones a los que se deben comparar los datos insinúan la infactibilidad de utilizar el esquema de apareamiento exacto a un conjunto de cadenas comunes. Las expresiones regulares han demostrado poseer mucha mayor flexibilidad y expresividad que las cadenas comunes al utilizarse en aplicaciones para redes de computadoras, particularmente para describir firmas de ataques (*attack signatures*) [15]. Por esto, se observa actualmente un auge en la utilización de expresiones regulares como patrones para este tipo de aplicaciones.

Las estructuras de datos y algoritmos analizados en este trabajo se sitúan dentro de este último enfoque, esto es, utilizan expresiones regulares como patrones para clasificar y controlar el tráfico, mientras que los algoritmos y estructuras se orientan hacia su utilización en dispositivos de red independientes.

Capítulo 4

AFD de entrada diferida

4.1 Motivación

Se sabe que la forma de representar una ER es a partir de la construcción de un autómata finito que reconozca el mismo lenguaje que la ER describe. Esto además puede deducirse de la equivalencia entre ambos expuesta en la sección 2.3.2. Resulta entonces claro que un sistema que utiliza ER como patrones para la inspección profunda de paquetes deberá implementar autómatas finitos de alguna forma u otra. Inicialmente existen dos opciones para hacer esto, utilizar AFDs o AFNs, de las cuales se debe elegir una. Después, como se va a trabajar con conjuntos de ER, se debe elegir entre construir un autómata independiente para cada ER o construir un solo autómata compuesto que reconozca todas las ER. Estas opciones se discuten a continuación:

AFD. En un autómata determinista, cada estado tiene una única transición para cada símbolo del alfabeto y además en cualquier momento del procesamiento se tiene un único estado activo. Estas dos observaciones indican que calcular el siguiente estado dada la lectura de un símbolo de la cadena de entrada puede lograrse en tiempo constante. Esto es, el tiempo de procesamiento para cada carácter de la entrada es $O(1)$. Ahora bien, como cada símbolo del alfabeto de entrada Σ genera una transición en cada estado, un autómata que representa

una expresión regular de n caracteres puede contener, en el peor caso, Σ^n estados. En aplicaciones para redes de computadoras, el alfabeto de entrada contiene los 256 símbolos del código ASCII extendido de 8 bits. Esto significa que en el peor de los casos, un autómata puede contener 256^n estados implicando un costo de almacenamiento de $O(\Sigma^n)$. En cuanto al número de autómatas que se pueden construir para representar un conjunto de m ER, se puede mantener la misma complejidad constante de procesamiento para un autómata compuesto, aunque esto incrementa la cantidad de estados a un peor escenario de $O(\Sigma^{nm})$, lo cual vuelve esta aproximación inviable. En cambio, construir m autómatas independientes incrementa el tiempo de procesamiento a $O(m)$ y los requerimientos de memoria, con un peor escenario de $O(m\Sigma^n)$, siguen siendo bastante altos.

AFN. Para los autómatas finitos no deterministas el panorama es distinto. Una expresión regular de longitud n puede representarse con un AFN cuyo número de estados esté acotado por $O(n)$, incluso en el peor de los casos. En cambio, el tiempo de procesamiento por cada carácter leído es muy alto, pues muchos estados pueden estar activos al mismo tiempo y todos ellos deben ser revisados. En el peor caso, en el cual todos los estados del autómata están activos, el tiempo de procesamiento es de orden $O(n^2)$. Para un autómata compuesto que representa un conjunto de m ER, el costo de almacenamiento alcanza una cota de $O(nm)$ mientras que la complejidad de procesamiento se incrementa, en el peor caso, a $O(n^2m)$. Puesto que lo que se intenta lograr es acelerar el tiempo de procesamiento, los AFN parecen no ser la opción adecuada para representar las ER.

4.2 Descripción

La estructura de datos presentada por Kumar et al. en [11] es una modificación a los AFD clásicos. Se busca con estas modificaciones mantener una complejidad de procesamiento baja al mismo tiempo que se reduce sustancialmente el costo de almacenaje de la estructura. Los llamados AFD de entrada diferida (o AFD^2) se construyen a partir de un AFD al alterar el conjunto de aristas para tratar de reducir lo más posible la cardinalidad de dicho conjunto, lo

cual representa un decaimiento en la cantidad de memoria necesaria para alojar la estructura. En esta sección, pensaremos en un AFD como la digráfica $G = (V, E)$ que lo representa en su diagrama de transición. Por tanto, tenemos que un autómata es una digráfica en la cual los vértices se llaman estados y las aristas dirigidas se llaman transiciones; uno de los vértices es designado como el estado inicial y para cada estado hay un conjunto, posiblemente vacío, de patrones que cazan. Las transiciones pueden o no estar etiquetadas por un símbolo de Σ. La noción primordial se obtiene del hecho de que si dos estados u y v de un AFD tradicional, al leer algún símbolo de un conjunto $C \subseteq \Sigma$ van a un mismo conjunto de estados $Q_{(u,v)} \subseteq Q$ entonces el autómata contiene información duplicada que se puede eliminar. Más formalmente, dado un AFD $M = (Q, \Sigma, q_0, \delta, F)$, si $u, v \in Q$ tienen la propiedad de que para un par de subconjuntos $C \subseteq \Sigma$ y $Q_{(u,v)} \subseteq Q$ sucede que $\forall a \in C,\ \delta(u, a) = \delta(v, a) \in Q_{(u,v)}$, entonces podemos agregar una transición *por omisión* de u a v (o de v a u) y modificar el conjunto de transiciones quitando todas las transiciones en común de ambos estados. Esta transición por omisión no está etiquetada en la digráfica y cada estado puede tener a lo más una de estas transiciones en su conjunto de transiciones salientes. Cuando un estado u está activo y se lee un símbolo a, se utiliza la transición (u, a) siempre que esta exista para cambiar de estado; si el autómata modificado no posee dicha transición, se cambia de estado siguiendo la transición por omisión de u sin consumir el símbolo de entrada y se repite el mismo procedimiento hasta lograr consumir el símbolo a al moverse por una transición etiquetada por él. Un autómata finito que contiene este tipo de transiciones se llama AFD de entrada diferida o AFD^2.

Tenemos ahora un autómata finito que continúa siendo determinista en el sentido de que en un mismo instante de tiempo sólo un estado puede estar activo y que para cada cadena w de símbolos de Σ hay un único camino que se puede seguir hasta su estado destino (se llama estado destino al estado que queda activo en el autómata después de leer uno a uno los símbolos de w y comenzando en el estado inicial q_0; se denota por $\delta(w)$ al estado destino de w aclarando que el camino que se sigue puede contener transiciones por omisión). El hecho de que el autómata continúe siendo finito nos garantiza un alto desempeño en tiempo de

procesamiento pues sólo se debe verificar un estado al leer un carácter de entrada, aunque ya no se hace en tiempo constante. El hecho de que no se consuma un símbolo al moverse por las transiciones por omisión resulta en un incremento en la complejidad de procesamiento que puede ser acotada superiormente por la longitud de la trayectoria dirigida más larga formada únicamente por transiciones sin etiqueta en el AFD^2 (llamaremos a estas últimas *trayectorias por omisión*). La construcción de un AFD de entrada diferida debe, a toda costa, evitar que se formen ciclos en la subgráfica dirigida formada por el conjunto de transiciones por omisión, pues esto puede significar la creación de ciclos en tiempo de ejecución. Además se puede restringir la longitud de las trayectorias por omisión para favorecer el desempeño en el procesamiento; esto se analiza a fondo más adelante.

4.3 Conversión y equivalencia

Este apartado consiste en demostrar la equivalencia entre un AFD y un AFD^2 construido a partir del primero para así poder validar su uso en aplicaciones prácticas. Se establecen primero algunas definiciones, en donde se supone que M, M_1 y M_2 son autómatas finitos deterministas.

Definición: *Dos autómatas M_1 y M_2 son* ***equivalentes*** *si y sólo si el lenguaje que ambos reconocen es el mismo, es decir, si $L(M_1) = L(M_2)$.*

Definición: *Una* ***configuración*** *de un autómata $M = (Q, \Sigma, q_0, \delta, F)$ es un elemento de $Q \times \Sigma^*$. La configuración (q, w) se refiere a que el estado q está activo en M y que falta por leer la cadena w. Por (q, aw) se denota la configuración en la cual q es el estado activo de M, aw es lo que resta de la cadena de entrada y a y w son un prefijo y un posfijo respectivamente de la cadena aw.*

Definición: *La* ***configuración inicial*** *para cualquier cadena $w \in \Sigma^*$ es (q_0, w).*

Definición: *La* ***relación paso de cálculo*** $\vdash$ *entre dos configuraciones se define como:*

$$((q_1, aw) \vdash (q_2, w)) \iff \delta(q_1, a) = q_2, \text{ o bien,}$$

$((q_1, aw) \vdash (q_2, aw)) \iff$ *existe una transición por omisión de* q_1 *a* q_2 *y* q_1 *no tiene ninguna transición saliente etiquetada por* a.

Se debe hacer notar que, dado el determinismo de los autómatas, en un solo paso de cálculo desde una configuración se puede pasar a una única configuración siguiente.

Definición: *La* ***relación proceso de cálculo*** $\vdash^*$ *entre dos configuraciones se define como:*

$((q_1, w) \vdash^* (q_2, x)) \iff$ *existe una sucesión de configuraciones* $C_0, C_1, \ldots, C_n$ *tales que* $C_0 = (q_1, w),\ C_n = (q_2, x)$ *y* $C_i \vdash C_{i+1}\ \forall\ 0 \leq i < n$.

Se observa que el lenguaje aceptado por un autómata M puede describirse ahora como:

$$L(M) = \{w \in \Sigma^* \mid (q_0, w) \vdash^* (q_f, \varepsilon),\ q_f \in F\}.$$

Se utilizan subíndices para especificar sobre qué autómata se desarrollan los pasos o procesos de cálculo cuando se pueda prestar a confusiones; si el entorno es claro se evita el uso de dichas referencias. Se enuncian en seguida algunos lemas para facilitar la demostración que esta sección pretende exponer. Sea M el AFD original y sea M_c el AFD^2 construido a partir del primero; se quiere demostrar que $L(M) = L(M_c)$.

Lema 1: *Sea* M_1 *un AFD*^{2} *con estados* u *y* v *tales que* $u \neq v$ *y* $\delta(u, a) = \delta(v, a)$ *para algún* $a \in \Sigma$*; además* u *no tiene ninguna transición por omisión saliente. Si se denota por* M_2 *al AFD*^{2} *construido a partir de* M_1 *al introducir una transición por omisión de* u *a* v *y quitar la transición de* u *a* $\delta(u, a)$*, se tiene que* M_1 *y* M_2 *son equivalentes.*

Demostración: *P.D.* $L(M_1) \subseteq L(M_2)$

Sea $w \in L(M_1)$*, por lo tanto existe una sucesión* C *de configuraciones que describen el proceso de cálculo* $(q_0, w) \vdash^*_{M_1} (q_f, \varepsilon)$ *para algún* $q_f \in F$*. Se entiende por* x *cualquier sufijo de*

w. Tenemos dos casos:

caso 1) Si (u, ax) no es una configuración contenida en la sucesión C, entonces la trayectoria seguida por w en M_1 no ha sido modificada en la construcción de M_2 por lo que se sigue esta misma trayectoria para llegar a q_f. $\therefore w \in L(M_2)$.

caso 2) Si (u, ax) es una configuración contenida en la sucesión C. Se tiene que en M, por cada presencia de la configuración (u, ax) se siguen necesariamente (por unicidad de los pasos de cálculo) los siguientes pasos de cálculo para consumir el símbolo a:

$$(u, ax) \vdash_{M_1} (\delta(u, a), x) \tag{4.1}$$

Mientras que en M_2 se siguen, por construcción, los siguientes pasos de cálculo:

$$(u, ax) \vdash_{M_2} (v, ax) \vdash_{M_2} (\delta(v, a), x) \tag{4.2}$$

Ahora bien, como $\delta_{M_1}(u, a) = \delta_{M_1}(v, a)$, la trayectoria en M_2 es la misma que en M_1 sustituyendo las transiciones de u a $\delta(u, a)$ por el par de transiciones mostrados en (4.2), esto es, la nueva transición por omisión de u a v y la transición de v a $\delta(v, a)$. Por lo tanto no se afecta el estado destino de $\delta(u, ax)$ para ninguna presencia de dicha configuración en C por lo que el estado destino de $\delta_{M_2}(q_0, w)$ en M_2 es el mismo que en M_1. Tenemos $(q_0, w) \vdash^*_{M_2} (q_f, \varepsilon)$ por lo que $w \in L(M_2)$. En cualquier caso se obtiene que $w \in L(M_2)$, $\therefore L(M_1) \subseteq L(M_2)$.

P.D. $L(M_2) \subseteq L(M_1)$
Sea $w \in L(M_2)$, existe una sucesión C de configuraciones que describen el proceso de cálculo $(q_0, w) \vdash^*_{M_2} (q_f, \varepsilon)$ para algún $q_f \in F$. Nuevamente se tienen dos casos:

caso 1) Como antes, si (u, ax) no es una configuración contenida en la sucesión C, la q_0q_f-trayectoria es la misma en ambos autómatas.

caso 2) Si la configuración (u, ax) aparece en C, entonces en M_2 para cada presencia se

siguen los pasos:

$$(u, ax) \vdash_{M_2} (v, ax) \vdash_{M_2} (\delta(v, a), x) \tag{4.3}$$

Mientras que en M_1 se siguen, por construcción, los siguientes pasos de cálculo:

$$(u, ax) \vdash_{M_1} (\delta(u, a), x) \tag{4.4}$$

Nuevamente, al ser $\delta_{M_1}(u, a) = \delta_{M_1}(v, a)$, las trayectorias que se siguen en ambos autómatas bajo w son las mismas salvo por las sustituciones en las transiciones que implican los pasos de cálculo en (4.3) y (4.4), por lo que no se afecta el estado destino de $\delta(u, ax)$ en ninguna de sus presencias en C. Tenemos entonces que $(q_0, w) \vdash^*_{M_1} (q_f, \varepsilon)$ por lo que $w \in L(M_1)$. En ambos casos se tiene $w \in L(M_1)$ por lo cual $L(M_2) \subseteq L(M_1)$.

De ambas contenciones se obtiene el resultado deseado, $L(M_1) = L(M_2)$, por lo que ambos autómatas son equivalentes.

□

***Lema 2** (generalización del Lema 1): Sea M_1 un AFD^2 con estados distintos u y v en los que u no tiene una transición por omisión saliente y que para un subconjunto S de Σ se tiene $\delta(u, a) = \delta(v, a)\ \forall a \in S$. Si se denota por M_2 al AFD^2 construido a partir de M_1 al introducir una transición por omisión de u a v y quitar la transición de u a $\delta(u, a)$, se tiene que M_1 y M_2 son equivalentes.*

***Demostración:** La demostración es directa del Lema 1 aplicando el mismo razonamiento para cada símbolo de S. La diferencia con el lema anterior reside en que en vez de sustituir una única transición, la inserción de la transición por omisión sustituye a todas las transiciones $\delta(u, a)$ en las que $a \in S$.*

□

El Lema 2 indica que el agregar transiciones por omisión válidas entre cualquier par de estados en un autómata puede reducir el número de transiciones al mismo tiempo que se preserva

la equivalencia en el autómata resultante. Para reducir los requerimientos de almacenaje de un autómata destinado a aplicaciones en redes de computadoras necesitamos reducir en gran medida el número de transiciones, por lo que la inserción de transiciones por omisión puede resultar muy útil para dicho fin. Sin embargo, necesitamos agregar más de una transición por omisión para obtener beneficios trascendentes; de aquí surge un cuestionamiento: ¿se pueden agregar varias transiciones por omisión y mantener la equivalencia con el autómata original?

Lema 3: *Sea M un AFD al que se le agregan sistemáticamente transiciones por omisión una tras otra siguiendo el procedimiento de construcción establecido en el Lema 2; entonces el AFD^2 resultante es equivalente a M.*

Demostración: *La demostración se obtiene por inducción aplicando el Lema 2 en cada paso. Es decir, como por cada transición por omisión nueva el Lema 2 garantiza la preservación de la equivalencia, entonces una aplicación repetida del mismo procedimiento resultará en un AFD^2 equivalente al AFD original.*

□

El Lema 3 deja en claro que se puede construir un AFD^2 a partir de un AFD con tantas transiciones por omisión como se desee; sin embargo, se debe considerar que estas transiciones se implementan para reducir las necesidades de memoria del autómata. Recordemos que solamente puede haber una transición por omisión saliente en cada estado, pues, como su nombre lo indica, será la transición que se utilizará por omisión en caso de que dicho estado no tenga una transición saliente etiquetada con el símbolo que se debe consumir. La presencia de más de una transición por omisión saliente en cualquier estado del autómata resultaría en la pérdida del determinismo, pues desde un mismo estado, leyendo un mismo símbolo del alfabeto de entrada, se podría arribar a estados diferentes (aquellos a los que apunten las distintas transiciones por omisión). Como segunda restricción, se requiere que las nuevas transiciones por omisión que se agreguen al autómata no formen ciclos compuestos únicamente de tran-

siciones sin etiquetar. Un ciclo formado exclusivamente por transiciones por omisión puede activar sucesivamente estados que rodeen dicho ciclo, entrando así en un proceso sin fin que además no consume un solo símbolo de la entrada, resultando en correspondientes ciclos de ejecución. Se deben entonces elegir las transiciones por omisión intentando maximizar el ahorro de memoria mientras se obedecen estas dos restricciones. Una observación más que surge al estudiar los lemas anteriores es que, al agregar transiciones por omisión, los estados finales del AFD^2 se pueden visitar sin haber leído una palabra del lenguaje que representa la expresión regular que aceptan dichos estados. Por lo tanto, los estados finales reportarán una presencia del patrón que aceptan solamente si son activados después de consumir un símbolo, es decir, si se llegó a ellos a través de una transición etiquetada.

Si se piensa nuevamente en un autómata como una digráfica G, al restringir la formación de ciclos formados únicamente por transiciones por omisión se observa que las aristas conformadas por dichas transiciones sin etiqueta componen una subgráfica acíclica de G, o lo que es lo mismo, un bosque. Se observa también que si se consideran dos estados u y v cuyas transiciones comparten destino para un número n de símbolos (i.e. $\delta(u,a) = \delta(v,a)\ \forall a \in C \subseteq \Sigma,\ |C| = n$), entonces una transición por omisión entre u y v eliminará n transiciones del autómata y simultáneamente agregará la transición sin etiqueta, dando como resultado una disminución de $n-1$ a la cardinalidad del conjunto de aristas. Con esto en mente, si se asigna a cada pareja de vértices de Q un peso, obtenido por la función

$$w\ :\ Q \times Q \to \mathbb{N}\ \cup\ \{0,-1\},$$

en la cual $w(u,v)$ es igual a uno menos del número de símbolos $a \in \Sigma$ distintos para los que se cumple $\delta(u,a) = \delta(v,a),\quad u,v \in Q,\ u \neq v$, el resultado es una gráfica completa (esto es con una arista entre cualesquiera dos vértices) y no dirigida, definida sobre el mismo conjunto de vértices que el autómata, en la que cada arista está etiquetada por el peso correspondiente a los dos vértices que inciden en ella. Se conoce a esta gráfica como la *gráfica de reducción*. Resulta entonces que la etiqueta de cada arista (u,v) en la gráfica de reducción representa la disminución en el número de aristas que se puede lograr insertando una transición por omisión entre los vértices u y v. Ahora bien, si se elige en la gráfica de reducción G_r de una

gráfica G cualquier subgráfica T, que a su vez sea un árbol generador de G_r (i.e. un árbol que contenga todos los vértices de G_r), y se elige un vértice de $V(T)$ para ser la raíz de dicho árbol, el peso total del árbol representará de la misma forma la cantidad de transiciones que se pueden eliminar si se agrega cada arista de T como una transición por omisión de G. Más aún, si se orientan estas transiciones por omisión hacia el vértice elegido como raíz de T, entonces se cumplen las restricciones para la inserción de trayectorias por omisión, como se demuestra en el siguiente lema.

Lema 4: *Las transiciones por omisión descritas por T y orientadas hacia la raíz de T obedecen las restricciones para la inserción de dichas transiciones en G.*

Demostración:

i) Libre de ciclos: por ser T un árbol, es acíclico, lo que garantiza que no se formarán ciclos entre las transiciones por omisión.

ii) Transiciones salientes únicas: queremos demostrar que ningún estado tendrá más de una transición por omisión saliente. Supongamos que existe un estado u en G que al agregar las trasiciones por omisión descritas por T resulta con más de una transición saliente sin etiqueta. Llámense v_1 y v_2 a dos de los vértices alcanzados por dos de estás transiciones por omisión salientes, que por estar estas últimas orientadas hacia la raíz indican que tanto v_1 como v_2 son padres de u. Por lo tanto, si denotamos por A_1 y A_2 a los respectivos conjuntos de aristas en las trayectorias de v_1 y v_2 a la raíz de T (bien definidos puesto que en un árbol estas trayectorias son únicas), entonces tenemos:

$$(A_1 \triangle A_2) \cup \{(u, v_1), (u, v_2)\} \text{ forman un ciclo en } T.$$

Esto es, la diferencia simétrica de A_1 y A_2 unida a las transiciones salientes de u forman un ciclo en T, lo cual contradice que T es acíclica; por lo tanto, no puede haber dos o más transiciones por omisión salientes de un mismo vértice.

□

Existen múltiples algoritmos para encontrar un árbol generador de peso máximo de una gráfica dada. Entre éstos se cuentan los algoritmos de Boruvka, de Prim y de Kruskal[10], en los cuales la subgráfica resultante representa la máxima reducción de aristas que se puede lograr al convertir un AFD a un AFD^2. De cualquier manera, en las aplicaciones a redes de computadoras estos árboles generadores de peso máximo, a pesar de reducir al mínimo la cantidad de memoria requerida para representar al AFD^2 correspondiente, suelen generar trayectorias por omisión demasiado largas. Recordando que al transitar por las trayectorias por omisión no se consume un solo símbolo de la entrada, la generación de muchas trayectorias largas de este tipo representan un aumento significativo en el tiempo de procesamiento, pues se requerirán tantos accesos a memoria como transiciones por omisión haya en la trayectoria para poder consumir un símbolo. Este aumento en la complejidad de procesamiento puede, inclusive, superar los beneficios obtenidos por la reducción en los requerimientos de memoria; por lo tanto es evidente que se debe acotar el tamaño de las trayectorias por omisión. La sección 5.5 se dedica a resolver este problema mientras que a continuación se ejemplifica la obtención de un AFD^2.

4.3.1 Ejemplo de AFD^2

Supongamos que se desea construir un AFD^2 que reconozca las expresiones regulares a^+, c^+b y $(a|b|c)^+d$. En primer lugar se debe construir un autómata finito determinístico mínimo que reconozca las tres expresiones. Se debe advertir que a diferencia de autómatas simples que sólo deciden si aceptan o no una palabra, este AFD debe reportar cualquier presencia de cualquiera de los patrones sin importar que cantidad de datos haya analizado antes. Es decir que resulta irrelevante qué estado se encuentre activo al comenzar a leer una subcadena de la entrada que sea reconocida por alguno de los patrones, pues dicha presencia debe reportarse. Por lo tanto la elección de un estado inicial es indistinta y este simplemente se especifica para poder comenzar la ejecución correctamente. Por esta razón, los diagramas de esta sección no especifican un estado inicial en ninguno de los autómatas. Retomando, se debe construir

un AFD mínimo que reconozca cualquier cadena representada por las expresiones regulares mostradas arriba. Este AFD se observa en la figura 4.1 dentro del cual el estado 1 reconoce la expresión a^+, el estado 4 reconoce a $(a|b|c)^+d$ y el estado 5 reporta las presencias de la expresión c^+b. A partir de este autómata *original*, podemos realizar el procedimiento para

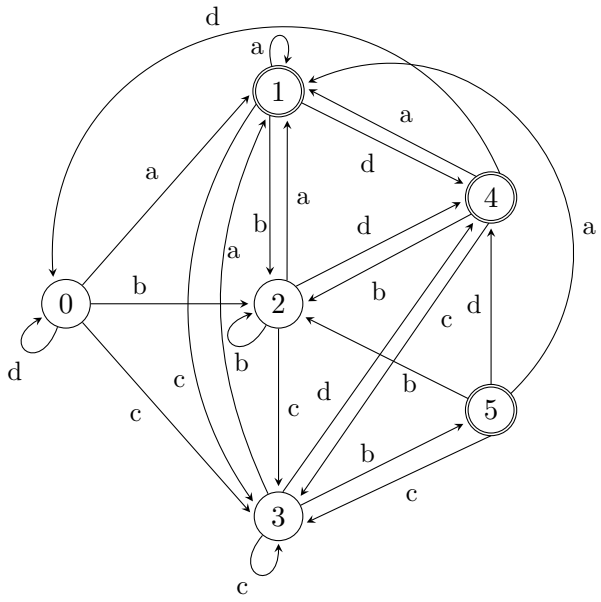

Figura 4.1: AFD que reconoce las ER: a^+, c^+b y $(a|b|c)^+d$.

agregar transiciones por omisión descrito arriba. Dependiendo de las parejas de estados que se elijan para generar dichas transiciones por omisión se obtendrán AFD^2 distintos. Se busca que al construir un AFD^2 se obtenga una compactación importante al reducir ampliamente el número de transiciones del autómata, así como mantener la longitud máxima de las trayectorias por omisión lo más pequeña que sea posible. Además se deben obedecer en todo momento las restricciones de no generar ciclos por omisión ni tener más de una transición saliente de este tipo desde ningún estado. Si se eligen las parejas de vértices (1,2), (1,3), (1,4), (1,5) y (0,4) para obtener las transiciones por omisión y se orientan hacia el nodo 1 las primeras cuatro y hacia el nodo 4 la última transición, se obtiene el siguiente AFD^2:

Este AFD^2 tiene solamente 11 transiciones, en contraste a las 24 del autómata original,

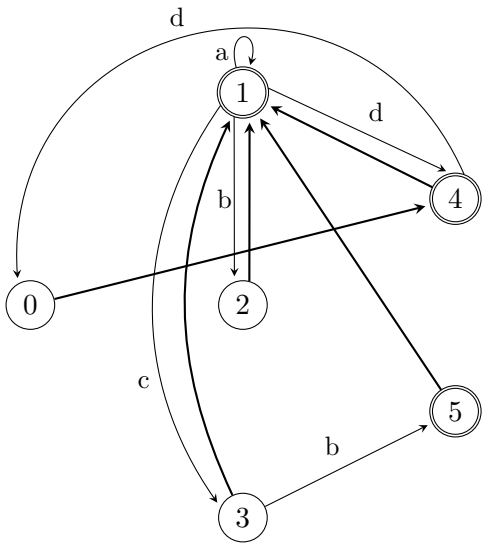

Figura 4.2: AFD^2 obtenido a partir del AFD original.

lo que representa una reducción mayor al 50%. Se ha visto que para gráficas de mayor complejidad utilizadas en aplicaciones a redes informáticas, esta disminución es incluso mayor, alcanzando en ocasiones una compactación superior al 90%. Sin lugar a dudas, el AFD^2 representa una importante mejora en relación a los requerimientos de memoria necesarios para alojar la estructura. Además, las trayectorias por omisión son relativamente cortas, con una longitud máxima de 2 en la trayectoria $0 \rightarrow 4 \rightarrow 1$ y una distancia promedio de 1 en las trayectorias por omisión. Es decir que si se supone un flujo de entrada con los símbolos de $\Sigma = \{a, b, c, d\}$ distribuidos aleatoriamente, en promedio se utilizará una transición por omisión para consumir cada símbolo.

Ahora bien, si al autómata de la figura 4.1 se le alimenta la cadena `ccbddad` comenzando en el estado 0, se seguirá la trayectoria $0 \rightarrow 3 \rightarrow 3 \rightarrow \bar{5} \rightarrow \bar{4} \rightarrow 0 \rightarrow \bar{1} \rightarrow \bar{4}$, en la cual los estados con una barra superior representan los estados finales. Es decir que se reporta una presencia del patrón c^+b al leer la primera `b`, en el siguiente símbolo se registra una presencia de $(a|b|c)^+d$ al activarse el estado 4, dos símbolos más tarde se encuentra

una presencia de a^+ y finalmente un reporte más por parte del estado 4. A su vez, si el AFD^2 que construimos consume esta misma cadena, la secuencia de estados que seguirá es $0 \to \bar{4} \to \bar{1} \to 3 \to \bar{1} \to 3 \to \bar{5} \to \bar{1} \to \bar{4} \to 0 \to \bar{4} \to \bar{1} \to \bar{1} \to \bar{4}$. Esto parece ser incorrecto pues se presentarían más presencias de los patrones en dicha trayectoria que símbolos en la entrada, por lo que se debe recordar que en los AFD^2 un estado final solamente reporta una presencia si se arriba a él por una transición etiquetada. Si reescribimos la trayectoria de estados que se sigue en el AFD^2 omitiendo los estados que son alcanzados por una transición por omisión (de manera que cualquier estado final que se presente implique el apareamiento de uno de los patrones), se obtiene la trayectoria $0 \to 3 \to 3 \to \bar{5} \to \bar{4} \to 0 \to \bar{1} \to \bar{4}$, que es exactamente la misma que la que se sigue en el autómata original (una consecuencia de los lemas 2 y 3). Por lo tanto, los resultados entregados por ambos autómatas son los mismos y

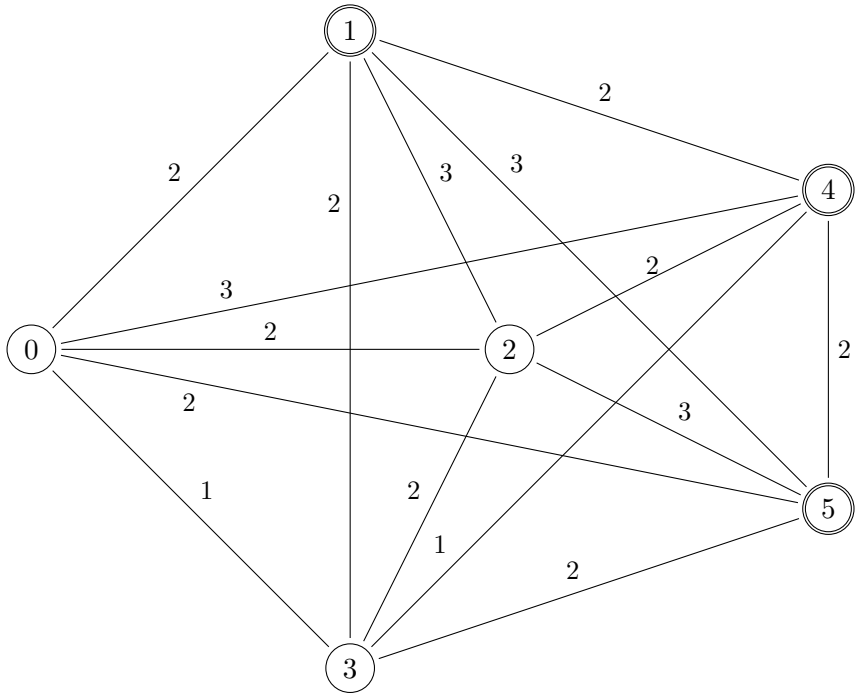

Figura 4.3: Gráfica de reducción para el autómata finito determinista.

las diferencias radican únicamente en el comportamiento interno y en las estructuras en sí. Como se mencionó arriba, una elección distinta de las parejas de vértices que definen las transiciones por omisión entregará AFD^2 diferentes que, a su vez, tienen distintas características.

Si se construye la gráfica de reducción del autómata finito original, como se muestra en la figura 4.3, se observa que el peso del árbol que definen las transiciones por omisión del AFD^2 sobre la gráfica de reducción es igual al número de transiciones de diferencia entre ambas estructuras[1].

Utilizando otras parejas de estados para definir las transiciones por omisión y orientando dichas transiciones como se muestra en la figura 4.4 se obtienen dos AFD^2 distintos para el mismo AFD original. Los AFD^2 mostrados en la figura 4.4 son estructuras con menor

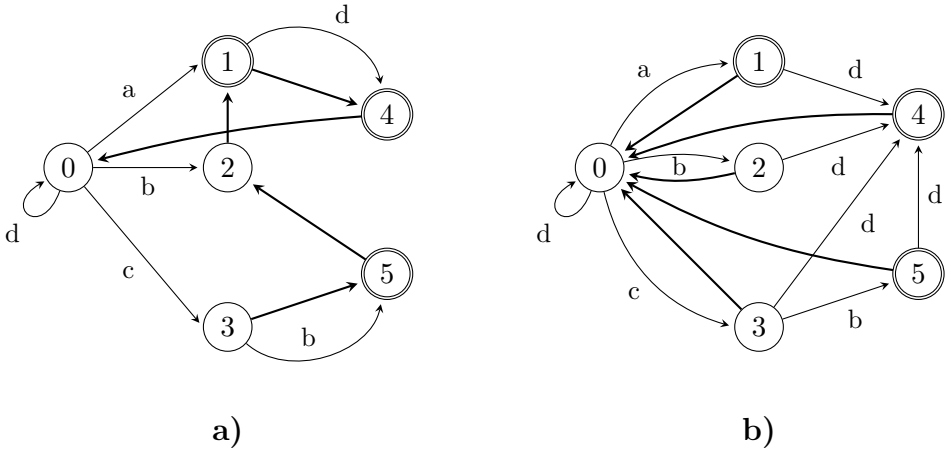

Figura 4.4: AFD^2 alternativos para el autómata original con distintas características.

desempeño que el primer AFD^2 que se obtuvo[2]. Esto ocurre por dos motivos independientes. Al igual que el primer AFD^2, el autómata de la figura 4.4a se obtiene a partir de un árbol generador de peso máximo de la gráfica de reducción, por lo cual la cantidad de transiciones que contiene es la mínima posible. Sin embargo, la orientación de las transiciones por omisión es desafortunada, resultando en una longitud máxima por omisión de 5 y una distancia

[1] En este caso las aristas (1, 2), (1, 3), (1, 4), (1, 5) y (0, 4) forman en la gráfica de reducción un árbol de peso $3 + 2 + 3 + 2 + 3 = 13$, que es en conecuencia la diferencia en el número de transiciones entre las 24 del AFD original y las 11 del AFD^2. Se puede además observar que para la gráfica de reducción (Figura 4.3) este generador árbol es de peso maximal, por lo que no se puede generar un AFD^2 con menos transiciones.

[2] Nuevamente, se sigue de los lemas 2 y 3 que ambos autómatas son equivalentes al AFD original pues reconocen el mismo lenguaje.

promedio por omisión de 2.5. Estos valores, mucho mayores a los del autómata de la figura 4.2, conllevan un importante incremento en el número de accesos a memoria necesarios para analizar un flujo de datos, afectando el desempeño en sus tiempos de ejecución. Por su parte, el autómata del lado derecho, figura 4.4b, se enfoca en minimizar la longitud de la trayectoria máxima por omisión. Aunque esta última estructura ciertamente reduce la distancia por omisión promedio de 1, en el primer AFD^2, a $\frac{5}{6}$, su desempeño en términos de memoria necesaria para ser alojado es un tanto menor, conteniendo un total de 14 transiciones. En autómatas más complejos, la diferencia que representa el aumentar el número de transiciones de 11 a 14, o en otras palabras, un incremento de 12.7%, puede resultar en requerimientos de memoria mucho mayores que limitarían la viabilidad de construir dispositivos independientes para el análisis de datos sobre los enlaces computacionales. Estos dos últimos AFD^2 muestran que se debe encontrar un balance óptimo entre la compactación y la velocidad de ejecución, esto se logra acotando la longitud de las trayectorias por omisión al tiempo que se intenta maximizar la compactación bajo ciertas restricciones. Este tema se discute a fondo en la siguiente sección.

4.4 AFD^2 con cotas sobre las trayectorias por omisión

La longitud de la trayectoria por omisión más larga en un AFD^2 sirve como una cota de desempeño para el peor caso, pues si la trayectoria de este tipo más larga contiene n aristas, se puede garantizar que se consumirá al menos un símbolo por cada n transiciones que se efectúen. Si se necesitan analizar grandes cantidades de datos en poco tiempo, se puede mejorar el rendimiento acotando la longitud de las trayectorias por omisión, pues se asegurará un mejor desempeño en el peor caso. Desgraciadamente, el limitar la longitud de las trayectorias por omisión puede ser un problema complicado pues los algoritmos clásicos dejan de funcionar para resolver este problema en la gran mayoría de los casos. Una primera aproximación para resolver este problema podría ser el tratar de encontrar un árbol de peso máximo acotando su

diámetro[3] a una cota máxima (no se puede pedir que el árbol sea generador pues esto puede no ser posible de conseguir). Sin embargo no es claro que el resultado de este procedimiento genere un AFD^2 mínimo o lo suficientemente pequeño; más aún, la construcción de árboles de peso máximo con cotas sobre la trayectoria máxima es un problema NP-duro con lo que cualquier motivación a persistir en esta aproximación se pierde de inmediato. Al perder toda viabilidad la construcción de un árbol generador único, la solución lógica es intentar construir un bosque generador de peso máximo sobre la gráfica de reducción, en el cual cada árbol tenga una cota sobre sus respectivos diámetros. Desgraciadamente, a pesar de esta modificación, la construcción del bosque planteado continúa siendo NP-duro para cualquier cota mayor a uno[4].

Para atacar el problema de acotar las trayectorias por omisión, Kumar et al.[11] han encontrado que un método basado en el agoritmo de Joseph Kruskal ofrece resultados notables sin aumentar en demasía la complejidad del preprocesamiento. La idea es muy similar al algoritmo para encontrar árboles generadores de peso máximo de Kruskal, que agrega las aristas de la gráfica al árbol a partir de sus pesos en orden decreciente, siempre que dicha arista no genere un ciclo. Adicionalmente, Kumar et al. agregan una restricción más a la adherencia de nuevas aristas, tomando en cuenta la longitud máxima de trayectorias. Los autores presentan dos versiones del algoritmo, de las cuales la segunda es un sencillo refinamiento del primer algoritmo que parece ofrecer beneficios importantes en las aplicaciones a redes de computadoras. La primera versión del algoritmo, al que denotaremos como NORM, se describe a continuación:

Algoritmo NORM (bosque generador *normal*):
Entrada: $G = (V, E)$ una gráfica y k una cota para la longitud de trayectorias.
Procedimiento:

[3] Recordemos que la trayectoria máxima en cualquier gráfica, en particular en los árboles, se conoce también como el diámetro de la gráfica.

[4] Puesto que para una cota con valor de 1, el problema se convierte en el de apareamientos de peso máximo, un clásico de la teoría de gráficas bastante simple de resolver

- Se crea un bosque B, donde cada vértice $v \in V(G)$ es un árbol.
- Se genera $\overline{E(G)}$ quitando de $E(G)$ las aristas con peso menor a 1.
- Mientras $\overline{E(G)} \neq \emptyset$
  - Se elige la arista $e \in \overline{E(G)}$ de peso máximo; en caso de haber más de una con el mismo peso, se elige cualquiera de ellas.
  - Si la arista e une dos árboles distintos y el diámetro (trayectoria máxima) en la unión de estos árboles es menor o igual que k, entonces se agrega e al bosque.
  - En otro caso se descarta la arista.
- Para cada árbol se elige un nodo raíz buscando minimizar la distancia de la raíz a sus hojas.

Salida: Bosque generador que cumple la restricción a la longitud máxima para trayectorias.

Sólo hay una diferencia medular entre el algoritmo de Kruskal para árboles generadores máximos y el algoritmo NORM, que se refiere a la verificación que se lleva a cabo para cada arista al analizar si el agregarla al bosque no infringe la propiedad de que no haya trayectorias más largas que la cota. Esta verificación se puede realizar usando una variable $d(u)$ para cada vértice u, que aloje la longitud de la trayectoria más larga desde u a cualquier vértice de su mismo árbol. Evidentemente, estas variables se deben actualizar para cada vértice dentro de la unión de árboles que genera la inserción de una nueva arista al bosque. Este último procedimiento se puede realizar en tiempo $O(n)$ cada que se agrega una nueva arista por lo que el procedimiento total de verificación de estas variables está acotado por $O(n^2)$. Ahora bien, el interés principal de este algoritmo es utilizarlo sobre las gráficas de reducción para decidir qué trayectorias por omisión se deben agregar al AFD^2, pero por construcción estas gráficas de reducción son gráficas completas. Al ser el algoritmo de Kruskal de orden $O(n^2 \log n)$ para gráficas completas, vemos que la complejidad asintótica sobre el tiempo de procesamiento del algoritmo NORM se mantiene igual al de Kruskal cuando se agrega el paso de verificación de trayectorias.

Se habló arriba de una segunda versión de este algoritmo, también presentada en Kumar et al [11], que representa un ligero refinamiento. Se denotará en lo subsiguiente a este

segundo algoritmo como REF (bosque generador con *refinamiento*) o simplemente como *refinamiento*. La idea es seguir el procedimiento del algoritmo NORM modificando la forma en que se eligen las aristas de $\overline{E(G)}$. En esta segunda versión del algoritmo las aristas se siguen eligiendo en orden decreciente de sus pesos y la alteración ocurre cuando se encuentra más de una arista con peso maximal dentro de $\overline{E(G)}$. Si este es el caso, en lugar de elegir cualquiera de ellas indistintamente, se elige aquella que represente el menor crecimiento en el diámetro del árbol (o árboles) que definen las transiciones por omisión. Ejemplificando con un caso muy simple, supóngase que se debe elegir entre, digamos, dos aristas e_1 y e_2, y se tiene que al agregar e_1 se obtendrá un nuevo árbol con diámetro m, mientras que al agregar a e_2 se obtendrá a un árbol con diámetro n; se elegirá agregar a e_1 en el escenario en que $m \leq n$ y se agregará a e_2 en el caso contrario. Es casi una garantía el hecho de que se encontrarán varias aristas del mismo peso durante la construcción de un AFD^2, pues como generalmente los protocolos sobre redes de computadoras utilizan el código ASCII como lenguaje de entrada, los valores posibles para los pesos de las aristas en $\overline{E(G)}$ son enteros entre 1 y 255, mientras que la cantidad de vértices en dichas gráficas suele estar en el orden de los miles, por lo que para muchos de esos pesos se tendrán varias aristas entre las cuales elegir. Se despliega a continuación el pseudocódigo para el refinamiento:

Algoritmo REF (bosque generador con refinamiento):

Entrada: $G = (V, E)$ una gráfica y k una cota para la longitud de trayectorias.

Procedimiento:

- Se crea un bosque B, donde cada vértice $v \in V(G)$ es un árbol.
- Se genera $\overline{E(G)}$ quitando de $E(G)$ las aristas con peso menor a 1.
- Mientras $\overline{E(G)} \neq \emptyset$
  - Se elige la arista $e \in \overline{E(G)}$ de peso máximo; en caso de haber más de una con peso máximo, se elige cualquiera que minimice el diámetro de la unión de los árboles. Es decir, para cada arista e de peso máximo, se elige aquella que minimice el crecimiento en el diámetro de los árboles que une; en caso de haber un empate en esta minimización, se elige cualquiera de ellas.
  - Si e no genera ciclos y el diámetro de la unión de los subárboles es menor o igual que k, entonces se agrega e al bosque.

  - En otro caso se descarta la arista.

- Para cada árbol se elige un nodo raíz buscando minimizar la distancia de la raíz a sus hojas.

Salida: Bosque generador que cumple la restricción a la longitud máxima para trayectorias.

Los resultados de Kumar et al. en [11], reportan que la diferencia sutil entre ambos algoritmos generalmente entrega diferencias importantes en su desempeño al utilizarlos en AFD que representan conjuntos de expresiones utilizados en aplicaciones a redes informáticas actuales. Afirman los autores que cuando no se especifica una cota para el diámetro de los árboles, ambos algoritmos generarán un árbol de peso máximo, pero el refinamiento entregará un árbol generador de relativamente menor diámetro (cuando esto sea posible) que el algoritmo original. Por otra parte, al especificar un límite para el diámetro de los árboles, el refinamiento no sólo entrega bosques cuyos árboles tienen menor diámetro en promedio, sino que también el bosque es de mayor peso que el que entrega el algoritmo NORM. Estos resultados experimentales parecen obedecer al hecho de que el algoritmo original genera árboles mucho más grandes que el refinamiento desde las primeras iteraciones, lo cual evita que pueda agregar muchas otras aristas de peso elevado más adelante puesto que las restricciones de diámetro no se lo permiten. En cambio, al mantener los diámetros reducidos al máximo, el refinamiento tiene mayores posibilidades de poder agregar esas aristas a la estructura.

Tomando en cuenta estas observaciones se idearon dos nuevos algoritmos para realizar esta tarea, los cuales se proponen y estudian a continuación. El primero de ellos estudia la percepción de que mientras más *estrellado*[5] sea el bosque generador, más aristas se podrán agregar al final pues los diámetros crecerán muy poco. Por lo que si a medida que se reducen los pesos, se elige al nodo que puede agregar más aristas válidas de dicho peso y se agregan todas esas aristas, se pueden lograr bosques generadores con un peso total bastante grande.

[5] Por estrellado nos referimos a la noción de redes con topología de *estrella*, en donde todos los nodos se conectan a un único nodo central

Se busca así implementar un algoritmo que contenga características de los dos algoritmos anteriores, al obtener la funcionalidad óptima del algoritmo NORM para algunos escenarios particulares, al mismo tiempo que se implementa un criterio de discriminación para elegir entre dos o más aristas en caso de un empate en los pesos. Puesto que dicho algoritmo busca en cada paso una maximización sobre la valencia de los nodos respecto a las aristas de la categoría de peso en la que se encuentra, recibe el nombre de algoritmo para maximización orientado a valencias (MOV). Mientras que para algunos escenarios extremos[6] el algoritmo entrega un desempeño de ejecución óptimo de $O(|E|) = O(n^2)$, un problema crucial surge del hecho que en gráficas generales es mucho más lento para ejecutarse que los demás algoritmos, debido al cambio en la forma de agregar aristas. En los algoritmos previos, se analizan las aristas una por una en orden decreciente de sus pesos y se decide si se agregan o no; en cambio ahora se observan los vértices y se analiza si se agrega un conjunto de aristas adyacentes a cada uno de ellos. Para ello, se deben examinar, para cada vértice, todas las aristas con peso igual al que se esté analizando y decidir cuántas de ellas es válido agregar en cada iteración; después se elegirá al vértice que pueda agregar más aristas comparando el crecimiento en las valencias de cada nodo. En el peor caso, se puede llegar a complejidades de ejecución de $O(n^3)$, en donde $n = |V|$, pues cada nodo puede requerir revisar cada una de las aristas para calcular el número de adyacencias válidas que puede agregar; dado que en una gráfica completa el número de aristas es $O(n^2)$ se obtiene la complejidad para el peor caso al mutliplicarla por el número de nodos. A continuación se muestra el pseudocódigo para el algoritmo MOV:

Algoritmo MOV (bosque generador por Maximización Orientada a Valencias):
Entrada: $G = (V, E)$ una gráfica y k una cota para la longitud de trayectorias.
Procedimiento:

- Se crea un bosque B, donde cada vértice $v \in V(G)$ es un árbol.
- Se genera $\overline{E(G)}$ quitando de $E(G)$ las aristas con peso menor a 1.

[6] Un ejemplo de estos escenarios extremos sería una gráfica completa (K_n) en la que todas las aristas tienen el mismo peso.

- $w = mayor\ peso\ posible$
- Mientras $w > 0$
  - Mientras se puedan agregar aristas
    - Se elige al nodo $v_{temp} \in V$ que pueda agregar el mayor número de aristas incidentes en él que cumplan que su peso sea igual a w y que además obedezcan las restricciones a la inserción de aristas en todo momento.
    - Se agregan las aristas válidas incidentes en v_{temp} que cumplen las restricciones al bosque generador.
  - $w \leftarrow w - 1$

Salida: Bosque generador que cumple la restricción a la longitud máxima para trayectorias.

El segundo algoritmo que se propone es una variación simplificada del algoritmo refinado (REF) expuesto en [11]. En REF, se busca minimizar el crecimiento de los diámetros en la unión de los subárboles que une cada arista válida, dichos valores se encuentran en el conjunto $\mathbb{N} \cup \{0\}$ por lo que el cero resulta la mejor minimización, es decir, no hay crecimiento en el diámetro de alguno de los subárboles al unirlo con el otro. Dicho en otra forma, si tenemos una arista $e = (u, v)$ donde u es un nodo del subárbol α y v un nodo del subárbol β y denotamos los diámetros de α y β como $diam(\alpha)$ y $diam(\beta)$, el crecimiento del diámetro al unir estos subárboles por e esta dado por:

$$incremento = max\left\{\ 0,\ d(u) + d(v) + 1 - max\{diam(\alpha), diam(\beta)\}\ \right\} \tag{4.5}$$

Por lo tanto, cuando se requiere de un criterio de desempate entre aristas con el mismo peso, el algoritmo REF debe analizar, para cada una de las aristas en cuestión, los dos subárboles y encontrar los respectivos diámetros, después debe realizar el cálculo de (4.5) y finalmente elegir aquella que minimice dicha expresión o bien la primera que entregue un incremento igual a cero. El segundo algoritmo que se propone busca simplificar los cálculos necesarios para discriminar las aristas más viables en caso de empates en los pesos. Por esta razón, se denota a dicho algoritmo como BGS (**B**osque **G**enerador **S**implificado). Este nuevo algoritmo

funciona de manera muy similar a REF con la diferencia de que en caso de empate en el peso de dos o más aristas, se agregará aquella que minimice la distancia de la trayectoria más larga que pase necesariamente por la arista analizada. Es decir que si tenemos para elegir n aristas válidas[7] de peso maximal, $e_1, e_2, \ldots, e_n$, en donde $e_i = (u_i, v_i)$ con $1 \leq i \leq n$, se agregará al bosque generador aquella e_i que minimice la expresión $d(u_i) + d(v_i) + 1$, pues ésta es la distancia de la trayectoria más larga que pasaría por cada una de las aristas e_i en caso de ser agregadas. Por lo tanto, sin importar si las aristas representan crecimiento en el diámetro de los subárboles que unen, se tiene un criterio para decidir cuál de ellas agregar si se tienen varias aristas de pesos iguales. Se debe notar que, a diferencia de REF, el algoritmo BGS no requiere información sobre los subárboles que las aristas elegidas unen, pues obtiene toda la información que necesita a partir de la tabla de distancias máximas desde cada nodo a otro de su mismo subárbol (esto es, la tabla de los valores $d(u)$). Se tiene entonces que, gracias a la simplificación del criterio de desempate, los tiempos de ejecución de BGS son mucho mejores que aquellos del algoritmo REF y en la práctica son de hecho casi tan buenos como los del algoritmo NORM. Así, el pseudocódigo del algoritmo BGS queda de la siguiente forma:

Algoritmo BGS (bosque generador simplificado):
Entrada: $G = (V, E)$ una gráfica y k una cota para la longitud de trayectorias.
Procedimiento:

- Se crea un bosque B, donde cada vértice $v \in V(G)$ es un árbol.
- Se genera $\overline{E(G)}$ quitando de $E(G)$ las aristas con peso menor a 1.
- Mientras $\overline{E(G)} \neq \emptyset$
  - Se elige la arista $e = (u, v) \in \overline{E(G)}$ de peso máximo; en caso de haber más de una con peso máximo, se elige cualquiera que minimice la expresión $d(u) + d(v) + 1$. Es decir, se elige alguna que minimice la distancia de la trayectoria más larga que pasará por ella una vez que se agregue al bosque generador. En caso de empate se elige cualquiera.
  - Si e no genera ciclos y el diámetro de la unión de los subárboles es menor o igual que k, entonces se agrega e al bosque.

[7] Por aristas válidas se entienden aquellas aristas que, al agregarse al bosque generador, no formen ciclos ni violen la cota a las trayectorias máximas.

  - En otro caso se descarta la arista.

- Para cada árbol se elige un nodo raíz buscando minimizar la distancia de la raíz a sus hojas.

Salida: Bosque generador que cumple la restricción a la longitud máxima para trayectorias.

Aunque el funcionamiento de este último algoritmo parece no preocuparse por mantener al mínimo el crecimiento de los árboles en el bosque generador, sí lo hace. El resultado de minimizar la longitud máxima de las trayectorias que atraviesan cada una de las aristas que se van agregando, termina por efectuar una importante minimización en el diámetro total de cada árbol, pues el diámetro es en sí una de las trayectorias que se han buscado minimizar al ir agregando las aristas que componen al árbol. Se muestra en el siguiente capítulo que para gráficas de reducción similares a las obtenidas de autómatas finitos que representan conjuntos de expresiones regulares utilizados en la industría, el algoritmo BGS entrega mejores resultados en cuanto a compresión que los otros tres algoritmos. Además, cuando se aplican cotas reducidas al diámetro máximo permisible, el AFD^2 que genera BGS tiene también un mejor desempeño promedio en tiempos de ejecución que los generados por los demás algoritmos. El resto de este capítulo se enfoca a describir el proceso de elección de raíces para los árboles contenidos en los bosques generadores de forma que se pueda optimizar el desempeño de los AFD^2.

4.5 Asignación de raíces

Una elección adecuada de las raíces en cada uno de los bosque del bosque generador es fundamental para el funcionamiento óptimo de los AFD^2, pues la distancia promedio de cada nodo en el bosque a su respectiva raíz sirve como una medida del número promedio de accesos a memoria que la estructura tendrá que realizar para consumir cada símbolo de la entrada. Por lo tanto, se debe elegir la raíz de cada árbol cuidadosamente para poder reducir este valor al mínimo. Se utiliza entonces como raíz al *centro* de cada árbol, definido a continuación, y se demuestran dos lemas relativos a las propiedades de dichos centros.

Definición: Sean $T = (V, E)$ un árbol y $v \in V$ un vértice en dicho árbol. Decimos que v es centro del árbol T si la distancia máxima de v a cualquier otro nodo del árbol es menor o igual que las trayectorias máximas de los demás vértices. Es decir,

$$v \text{ es centro de } T \iff d(v) \leq d(x)\ \forall x \in V$$

En los siguientes lemas se utiliza la notación $d(u, v)$ para denotar la longitud de la trayectoria entre los vértices u y v. Puesto que se trabaja con árboles, dichas trayectorias son únicas y los valores están, en consecuencia, bien establecidos. A su vez, la expresión $d(v)$ mantiene el significado que se maneja a lo largo de este trabajo y se refiere a la longitud de la trayectoria más larga desde el nodo v a cualquier otro nodo de su gráfica.

Lema 5: Si un vértice v es centro de un árbol T, entonces v está contenido en el diámetro de T.

Demostración: Sea T un árbol cuyo diámetro sea la trayectoria de un vértice x a un vértice y (llamemos a esta trayectoria D) y centro en un vértice v. Supongamos que v no está en el diámetro de T. Por ser T un árbol, es conexo, por lo que existe una única trayectoria de v a x, denotemos por z al primer vértice contenido en el diámetro D que intersecta con la vx-trayectoria si se empieza recorriendo desde v. Definamos las siguientes distancias:

- Sea $a = d(v, z)$.
- Sea $k = max\{d(z, x), d(z, y)\}$

Tenemos que $k \geq d(z, w)\ \forall w \in V(T)$, pues de otra forma existiría una trayectoria más larga que D (explícitamente, alguna de las trayectorias $uz \cup zx$ o bien $uz \cup uy$ sería más larga que D donde u es cualquier vértice de $V(T)$ que maximice la expresión $d(z, u)$). $\therefore d(z) = k$. Ahora bien $max\{d(v, x), d(v, y)\} = k + a \implies d(v) > d(z)$ **!**
Por lo tanto v debe estar en el diámetro de T.

□

Lema 6: Un árbol tiene a lo más dos centros.

Demostración: Sabemos del lema anterior que cualquier centro de un árbol debe estar contenido en el diámetro del mismo. Sea el diámetro una trayectoria de un vértice x a un vértice y, entonces los centros, o vértices que minimizen la expresión $max\{d(z,x), d(z,y)\}$, deben estar situados hacia la mitad del diámetro. Es fácil observar que si el número de nodos en el diámetro es par habrá dos centros en el árbol y si el número de nodos en el diámetro es impar sólo existirá uno.

□

Con los lemas anteriores en mente se define el método para elegir las raices de los árboles del bosque generador. Puesto que ya se utiliza una tabla que guarda los valores $d(v)$ para cada nodo en la gráfica, simplemente se debe buscar el vértice de cada árbol que minimice este valor. Esto se puede hacer en tiempo lineal para todas las componentes conexas en el bosque por lo que la elección de raíces no cambia el desempeño de ninguno de los algoritmos NORM, REF, MOV o BGS. En caso de que haya dos centros para cualquiera de los árboles, se elige como raíz aquel con mayor valencia. Además, el utilizar el mismo método para la elección de raíces en todos los algoritmos da validez a los resultados de los estudios comparativos presentados en el siguiente capítulo.

Capítulo 5

Estudios comparativos

5.1 Definición de tipos de gráficas

Durante los análisis y pruebas se utilizaron ampliamente dos estilos de gráficas con diferentes características. Se diferenciaron escencialmente dos tipos que se denotan en lo sucesivo como gráficas de tipo A y B, cada una con sus propias particularidades que se establecen a continuación. Estas gráficas a su vez, representan las gráficas de reducción generadas sintéticamente sobre las cuales se ejecutaron los algoritmos de compactación. Las características de las gráficas son simplemente propiedades generales, pues la generación de cada gráfica de reducción incluye procesos aleatorios que practicamente imposibilitan la generación de dos gráficas iguales. Por lo tanto, para las pruebas con cada uno de los tipos de gráfica, los algoritmos se corren sobre un conjunto de gráficas con pesos y asignaciones muy diferentes, aunque con características generales similares.

Gráficas tipo A. Se codificó un programa que genera un AFD sintético con el número de estados que se le solicita y asigna transiciones entre ellos, la asignación de transiciones sigue el mismo método propuesto en [11]. Dicho método consiste en que a cada par de estados u y v se les asigna un número aleatorio de símbolos $a \in \Sigma$ tales que cumplan que $\delta(u, a) = \delta(v, a)$[1]. El número aleatorio se obtiene a partir de una distribución geométrica con probabilidad de

[1] En donde δ denota a la función de transición del autómata.

éxito de $\pi = 0.05$ y acotando los valores resultantes al conjunto $\{n \in \mathbb{N} \cup \{0\} \mid 0 \leq n \leq 255\}$. De esta forma se obtiene fácilmente la gráfica de reducción y los pesos de las aristas que unirán los vértices en ella, los que a su vez, están también distribuidos geométricamente. Esto es, el peso de la arista entre dos vértices cualesquiera en la gráfica de reducción será 0 con probabilidad 0.05 (lo que es lo mismo, los vértices unidos comparten una sola transición, por lo que no se agregará dicha arista a la gráfica de reducción), será 1 con probabilidad 0.0475, 2 con probabilidad 0.45125 y en general será n con probabilidad

$$P(n) = (1 - \pi)^n \pi$$

en donde, nuevamente, $\pi = 0.05$. Denotaremos las gráficas construidas de esta forma como *gráficas tipo A*.

Gráficas tipo B. Las particularidades de este segundo estilo de gráfica se recolectaron a partir de un análisis de los AFD utilizados por Kumar et. al. [11] en sus pruebas, los cuales incluyen conjuntos de expresiones regulares utilizados en sistemas de detección de intrusiones en redes tales como Snort, Bro, L7 y Cisco. También se consideran las apreciaciones de Antonatos et. al. [3] en su artículo sobre la generación de datos para sistemas de detección de intrusiones. Se estudiaron aspectos como la proporción de transiciones duplicadas en el AFD, la proporción de ciclos en la gráfica de reducción correspondiente, los valores de los pesos en las aristas de la gráfica de reducción y la longitud promedio de las trayectorias por omisión que se generan en el AFD[2] al producir los bosques generadores sin la aplicación de cotas. A partir de estas observaciones se implementó un nuevo programa que construye gráficas de reducción sintéticas con todas las características necesarias para representar de manera apropiada un AFD representativo de los utilizados en las aplicaciones a redes informáticas; denotamos a las gráficas con estas propiedades como *gráficas tipo B*. En primer lugar, se obtuvo un porcentaje constante de aristas duplicadas en el autómata finito al promediar dicho porcentaje en las pruebas estudiadas; éste resultó ser de 97.77%[2], por lo que todas

[2] Este valor a su vez representa una cota superior a la máxima compactación posible que se puede lograr en el autómata finito sin perdida de información.

las gráficas de reducción tipo B presentan esta proporción. Para construir las gráficas de reducción sintéticas se le especifica al programa el número de transiciones duplicadas que se tendrían en el AFD original y apartir de este número, la aplicación calcula el número total de transiciones en el AFD así como una cota para el peso total de la gráfica de reducción, limitada por 1.02 veces el número de aristas duplicadas. Esta constante multiplicativa surge a partir del hecho de que en los estudios analizados previamente, la proporción de transiciones originales que dan lugar a las duplicadas en las gráficas de reducción suele ser alrededor de una a cincuenta, de aquí que el peso total de la gráfica de reducción tenga un aumento de alrededor del 2% superior al número de transiciones duplicadas en el AFD. Recordemos que siempre que exista un conjunto de dos o más transiciones tales que $\delta(v_1, a) = \delta(v_2, a) = \ldots = \delta(v_n, a)$ con $v_i \neq v_j \ \forall i \neq j,\ a \in \Sigma$, una de las transiciones se denota como la *original* mientras que todas las demás se denotan como transiciones *duplicadas*. Como las transiciones originales, además de todas las duplicadas, se deben consideran al momento de asignar pesos a las aristas en la gráfica de reducción y la proporción entre originales y duplicadas es de 1:50, se agrega ese porcentaje extra a la gráfica de reducción. También, basado en las observaciones, se permite que un número de aristas que se agregan a la gráfica generen ciclos en ella; este número a pesar de ser aleatorio se mantiene muy cercano a la proporción observada en los demás estudios de 15%. Otro detalle que se observó en las gráficas de reducción analizadas es que el promedio de peso de cada arista utilizada en los bosques generadores era muy cercano al peso máximo posible de 255 (bajo la utilización de ASCII) por lo que la asignación de pesos utilizada en los primeros experimentos no produce los resultados deseados. Con esto en cuenta se modificó la función de asignación de pesos aleatorios, se utilizó de nueva cuenta una distribución geométrica pero ahora con probilidad de éxito de $\pi = 1/3$ y por tanto media esperada de 2. Además, en lugar de asignar los valores obtenidos de la distribución como los pesos de las aristas, se restan dichos valores de 255 para obtener la asignación de peso. Así, la media estadística del peso de las aristas es de 253.

Para verificar los resultados reportados por Kumar et al. [11] y comparar el desempeño de los nuevos algoritmos, se programaron los algoritmos NORM, REF, MOV y BGS en el

lenguaje Java y se realizaron múltiples experimentos analizando varias características de los AFD^2 que los algoritmos entregaban. Las próximas cuatro secciones se enfocan en describir y analizar un estudio comparativo entre los algoritmos NORM, REF y BGS al ejecutarse sobre ambos tipos de gráficas. La comparación del desempeño del algoritmo MOV se deja hacia el final del capítulo pues dicho algoritmo no presenta ninguna mejoría tangible frente a los demás y por tanto se incluyen los análisis sobre este algoritmo simplemente como testimonio de su inviablidad práctica.

5.2 Ejecuciones sobre gráficas de reducción tipo A

Se generaron varias gráfica de reducción de diferentes tamaños y sobre ellas se corrieron los tres algoritmos NORM, REF y BGS con diferentes cotas para comparar su desempeño. Cada algoritmo obtiene su respectivo bosque generador intentando maximizar el peso y limitando los diámetros de los árboles a la cota solicitada. Después de producir el bosque generador resultante de la gráfica de reducción y la limitante de tamaños, cada uno de los tres algoritmos, elige las raíces óptimas para cada árbol dentro del bosque para poder evaluar apropiadamente el desempeño del AFD^2. Esto es, el desempeño que tendría la estructura en caso de utilizar dicho bosque generador de la gráfica de reducción para asignar las transiciones por omisión. El peso total del bosque generador se comprende como el ahorro en el número de transiciones que significará el AFD^2 con respecto al AFD, mientras que la distancia promedio a la raíz definirá la mejoría en el número de accesos a memoria necesarios para consumir cada carácter del flujo de entrada. Es decir que el peso total del bosque generador representa la ganancia que ofrece el AFD^2 en términos de memoria, mientras que la distancia promedio a la raíz representa la mejoría en cuanto a los tiempos de ejecución.

Se utilizaron gráficas de reducción de diferentes órdenes con 20, 30, 60, 100, 300, 600 y 1000 nodos, y para cada uno de estos órdenes se utilizaron diferentes cotas. El conjunto total de cotas comprende los valores 2, 4, 7, 10, 20, 30 e ∞, en donde el valor infinito representa el escenario en el que no se acota el crecimiento de los árboles en el bosque generador. Se debe

mencionar que se evitaron los casos en los que un valor real[3] de acotamiento fuera superior al número de nodos de la gráfica pues dichos casos se encuentran ya representados en el caso sin acotar, es decir, cuando el valor de la cota es ∞. Además, para cada combinación válida posible de número de nodos y cota se realizaron cien ejecuciones de los algoritmos para obtener los valores promedio que se observan en los resultados, contenidos en la Tabla 5.1.

Cada una de las ejecuciones genera una gráfica de reducción diferente asignando pesos a las aristas aleatoriamente con el procedimiento de distribuciones geométricas explicado arriba, por lo que los valores promedio mostrados en la Tabla 5.1 son altamente confiables como valores de desempeño general de los algoritmos NORM, REF y BGS sobre gráficas de tipo A. Para cada escenario orden-cota se obtuvieron diversos datos como son el peso promedio de los bosques generadores a través de las múltiples iteraciones, la distancia promedio a la raíz de cada nodo en cada bosque generador, la media de la distancia máxima por omisión en cada bosque generador (i.e. la distancia máxima de un nodo a otro nodo de su mismo árbol en cada árbol generador), el promedio de la valencia máxima de cada bosque generador y finalmente el número promedio de aristas agregadas a cada uno de los bosques. Cada uno de estos datos se obtiene para los tres algoritmos corriendo sobre el mismo conjunto de cada bosque generador, se decide qué algoritmo representa una mayor de gráficas de reducción. Además, con los pesos totales individuales reducción al número de aristas y se lleva un marcador entre los tres algoritmos considerando las cinco situaciones posibles: número de veces en que NORM representa una mejor compactación que los otros dos, número de veces en que REF resulta mejor, aquellas en las que BGS es mejor, número de veces en que la reducción de los tres algoritmos es igual y la cantidad de veces en las que se da un comportamiento distinto. Un ejemplo de comportamiento distinto es cuando REF es mejor NORM, BGS es mejor que NORM y BGS compacta en la misma proporción que REF. Evidentemente, la suma de los cinco valores de este marcador equivale al número de ejecuciones para cada escenario, como se constata en la Tabla 5.1.

[3] i.e. sin contar el valor ∞.

Tabla 5.1: Resultados en gráficas tipo A.

			Marcador					PesoTotal	AristasTot.	Peso Promedio			Dist Prom a Raíz		
Nodos	Cota	It.	NORM	REF	BGS	Igual	Cíclico	estad.	estad.	NORM	REF	BGS	NORM	REF	BGS
1000	∞	100	0	0	0	100	0	9015975	474525	138237.78	138237.78	138237.78	16.769	16.127	15.838
1000	30	100	13	42	28	3	14	9015975	474525	137806.04	137828.24	137801.49	8.8802	8.9587	8.7088
1000	20	100	23	48	20	2	7	9015975	474525	137234.17	137277.29	137240.76	6.0187	6.0375	5.926
1000	10	100	14	60	22	0	4	9015975	474525	133927.6	133996.3	133921.7	3.1116	3.1195	3.0891
1000	7	100	11	76	11	0	2	9015975	474525	130027.44	130128.99	130001.42	2.2287	2.2357	2.2223
1000	4	100	15	72	8	0	5	9015975	474525	119805.79	119931.23	119705.26	1.3348	1.3369	1.3291
1000	2	100	3	78	0	0	19	9015975	474525	97342.2	97462.77	97119.35	0.7063	0.7075	0.704
600	∞	100	0	0	0	100	0	3243585	170715	77078.17	77078.17	77078.17	13.769	13.192	12.979
600	30	100	11	26	22	18	23	3243585	170715	76807.46	76821.44	76813.24	8.7886	8.8334	8.6763
600	20	100	14	38	22	10	16	3243585	170715	76429.97	76450.52	76419.67	5.9506	5.9724	5.876
600	10	100	17	54	20	1	8	3243585	170715	74435.92	74472	74414.68	3.1021	3.11	3.077
600	7	100	9	72	10	3	6	3243585	170715	72220.35	72280.52	72214.44	2.2313	2.2371	2.2204
600	4	100	14	61	12	3	10	3243585	170715	66726	66792.48	66673.86	1.3354	1.3372	1.3291
600	2	100	1	58	4	7	30	3243585	170715	54144.87	54207.14	54022.4	0.7054	0.7065	0.7031
300	∞	100	0	0	0	100	0	809543	42608	34381.34	34381.34	34381.34	10.345	10.143	10.034
300	30	100	8	6	16	48	22	809543	42608	34355.24	34357.17	34356.84	8.4892	8.512	8.3941
300	20	100	4	22	22	23	29	809543	42608	34098.89	34111.82	34104.92	5.9421	5.9496	5.8754
300	10	100	7	41	21	14	17	809543	42608	33259.93	33278.41	33253.83	3.093	3.1042	3.0674
300	7	100	11	37	17	12	23	809543	42608	32389.16	32405.12	32384.42	2.2287	2.2351	2.2244
300	4	100	10	48	11	12	19	809543	42608	29798.02	29829.27	29780.69	1.331	1.3342	1.3255
300	2	100	0	34	8	23	35	809543	42608	24136.04	24160.82	24077.94	0.7058	0.7069	0.7031
100	∞	100	0	0	0	100	0	89348	4703	9262.61	9262.61	9262.61	6.3405	6.2016	6.1538
100	30	100	2	0	0	95	3	89348	4703	9308.44	9308.68	9308.68	6.5352	6.3686	6.3231
100	20	100	2	5	6	70	17	89348	4703	9212.77	9215.43	9214.82	5.568	5.531	5.4703
100	10	100	4	9	11	55	21	89348	4703	9008.46	9009.74	9007.16	3.088	3.0819	3.0592
100	7	100	6	16	6	48	24	89348	4703	8763.99	8767.82	8757.29	2.2196	2.2273	2.2155
100	4	100	2	22	7	51	18	89348	4703	8054.12	8062.44	8051.09	1.3295	1.3341	1.3246
100	2	100	0	13	2	67	18	89348	4703	6557.87	6563.34	6548.21	0.7052	0.7061	0.7031
60	∞	100	0	0	0	100	0	31949	1682	4918.45	4918.45	4918.45	5.0692	5.0108	4.9785
60	30	100	0	0	0	100	0	31949	1682	4915.67	4915.67	4915.67	4.9392	4.8387	4.832
60	20	100	0	0	0	97	3	31949	1682	4948.15	4948.26	4948.26	4.9015	4.8445	4.816
60	10	100	2	8	5	71	14	31949	1682	4833.78	4835.75	4833.49	3.0853	3.0818	3.0655
60	7	100	3	10	4	65	18	31949	1682	4690.34	4692.4	4687	2.1882	2.1958	2.177
60	4	100	2	14	3	67	14	31949	1682	4324.65	4329.04	4319.08	1.341	1.3453	1.3347
60	2	100	1	7	4	77	11	31949	1682	3493.06	3496.19	3486.96	0.7022	0.7032	0.6997

Tabla 1.1: Resultados en gráficas tipo A

		Dist Máx×Omisión			Valencia Máx Prom			Aristas Promedio			% aristas usadas			% peso total		
Nodos	Cota	NORM	REF	BGS	NORM	REF	BGS	NORM	REF	BGS	NORM	REF	BGS	NORM	REF	BGS
1000	∞	33.49	31.93	31.75	7.06	7.07	7.1	999	999	999	0.211%	0.211%	0.211%	1.533%	1.533%	1.533%
1000	30	15	15	15	7.22	7.25	7.28	997.06	997.35	997.1	0.210%	0.210%	0.210%	1.528%	1.529%	1.528%
1000	20	10	10	10	7.43	7.43	7.38	992.79	993.07	992.67	0.209%	0.209%	0.209%	1.522%	1.523%	1.522%
1000	10	5	5	5	7.06	7.12	7.14	970.49	971.1	970.26	0.205%	0.205%	0.204%	1.485%	1.486%	1.485%
1000	7	4	4	4	7.29	7.4	7.27	945.1	946.05	944.9	0.199%	0.199%	0.199%	1.442%	1.443%	1.442%
1000	4	2	2	2	7.08	7.19	7.07	871.64	872.77	870.45	0.184%	0.184%	0.183%	1.329%	1.330%	1.328%
1000	2	1	1	1	6.54	6.58	6.4	706.32	707.52	704	0.149%	0.149%	0.148%	1.080%	1.081%	1.077%
600	∞	27.02	25.87	25.88	6.78	6.77	6.86	599	599	599	0.351%	0.351%	0.351%	2.376%	2.376%	2.376%
600	30	15	15	15	6.82	6.85	6.86	598.1	598.22	598.1	0.350%	0.350%	0.350%	2.368%	2.368%	2.368%
600	20	10	10	10	6.91	6.91	6.89	595.59	595.75	595.43	0.349%	0.349%	0.349%	2.356%	2.357%	2.356%
600	10	5	5	5	6.88	6.93	6.89	582.2	582.53	581.83	0.341%	0.341%	0.341%	2.295%	2.296%	2.294%
600	7	4	4	4	6.75	6.82	6.84	567	567.58	566.82	0.332%	0.332%	0.332%	2.227%	2.228%	2.226%
600	4	2	2	2	6.58	6.67	6.59	523.25	523.9	522.52	0.307%	0.307%	0.306%	2.057%	2.059%	2.056%
600	2	1	1	1	6.25	6.28	6.17	423.22	423.88	421.86	0.248%	0.248%	0.247%	1.669%	1.671%	1.666%
300	∞	20.39	19.86	19.84	6.39	6.4	6.41	299	299	299	0.702%	0.702%	0.702%	4.247%	4.247%	4.247%
300	30	14.92	14.92	14.91	6.45	6.46	6.46	298.98	298.98	298.97	0.702%	0.702%	0.702%	4.244%	4.244%	4.244%
300	20	10	10	10	6.39	6.48	6.43	297.64	297.76	297.69	0.699%	0.699%	0.699%	4.212%	4.214%	4.213%
300	10	5	5	5	6.49	6.53	6.49	290.9	291.12	290.79	0.683%	0.683%	0.682%	4.108%	4.111%	4.108%
300	7	4	4	4	6.55	6.53	6.52	283.42	283.61	283.39	0.665%	0.666%	0.665%	4.001%	4.003%	4.000%
300	4	2	2	2	6.32	6.36	6.28	261.4	261.8	261.08	0.614%	0.614%	0.613%	3.681%	3.685%	3.679%
300	2	1	1	1	5.69	5.78	5.62	211.75	212.08	210.94	0.497%	0.498%	0.495%	2.981%	2.985%	2.974%
100	∞	12.1	11.77	11.78	5.59	5.6	5.6	99	99	99	2.105%	2.105%	2.105%	10.367%	10.367%	10.367%
100	30	12.32	12.04	12.02	5.51	5.52	5.55	99	99	99	2.105%	2.105%	2.105%	10.418%	10.419%	10.419%
100	20	9.92	9.85	9.85	5.64	5.65	5.63	98.99	99	98.99	2.105%	2.105%	2.105%	10.311%	10.314%	10.313%
100	10	5	5	5	5.81	5.82	5.79	97.07	97.1	97.06	2.064%	2.065%	2.064%	10.082%	10.084%	10.081%
100	7	4	4	4	5.63	5.68	5.61	94.32	94.39	94.26	2.006%	2.007%	2.004%	9.809%	9.813%	9.801%
100	4	2	2	2	5.39	5.46	5.38	87.03	87.21	86.95	1.851%	1.855%	1.849%	9.014%	9.024%	9.011%
100	2	1	1	1	5	5.03	4.97	70.52	70.61	70.31	1.500%	1.502%	1.495%	7.340%	7.346%	7.329%
60	∞	9.43	9.3	9.3	5.18	5.18	5.22	59	59	59	3.509%	3.509%	3.509%	15.395%	15.395%	15.395%
60	30	9.35	9.13	9.15	5.22	5.22	5.24	59	59	59	3.509%	3.509%	3.509%	15.386%	15.386%	15.386%
60	20	9.1	8.97	8.97	5.18	5.21	5.24	59	59	59	3.509%	3.509%	3.509%	15.488%	15.488%	15.488%
60	10	5	5	5	5.39	5.38	5.35	58.2	58.21	58.18	3.461%	3.462%	3.460%	15.130%	15.136%	15.129%
60	7	4	4	4	5.26	5.28	5.26	56.49	56.54	56.44	3.360%	3.362%	3.357%	14.681%	14.687%	14.670%
60	4	2	2	2	5.32	5.34	5.26	52.47	52.56	52.35	3.120%	3.126%	3.113%	13.536%	13.550%	13.519%
60	2	1	1	1	4.62	4.64	4.54	42.13	42.19	41.98	2.506%	2.509%	2.497%	10.933%	10.943%	10.914%

Tabla 1.1: Resultados en gráficas tipo A

			Marcador					PesoTotal	AristasTot.	Peso Promedio			Dist Prom a Raíz		
Nodos	Cota	It.	NORM	REF	BGS	Igual	Cíclico	estad.	estad.	NORM	REF	BGS	NORM	REF	BGS
30	∞	100	0	0	0	100	0	7852	413	2016.14	2016.14	2016.14	3.389	3.3177	3.3123
30	20	100	0	0	0	100	0	7852	413	2028.51	2028.51	2028.51	3.514	3.469	3.46
30	10	100	2	2	6	87	3	7852	413	2004.8	2004.78	2004.45	2.9583	2.9597	2.944
30	7	100	2	5	3	83	7	7852	413	1955.52	1956.37	1955.12	2.1633	2.1787	2.16
30	4	100	2	6	1	79	12	7852	413	1796.16	1797.12	1793.57	1.3127	1.3147	1.304
30	2	100	0	1	3	90	6	7852	413	1464.72	1465.09	1463.09	0.7023	0.7027	0.
20	∞	100	0	0	0	100	0	3430	181	1178.3	1178.3	1178.3	2.7975	2.768	2.757
20	10	100	1	0	1	95	3	3430	181	1145.38	1145.57	1145.56	2.6945	2.6875	2.679
20	7	100	1	2	3	92	2	3430	181	1145.07	1145.71	1145.31	2.153	2.157	2.136
20	4	100	0	2	0	94	4	3430	181	1073.56	1074.07	1073.06	1.3345	1.333	1.32
20	2	100	0	2	2	89	7	3430	181	861.78	862.25	860.1	0.701	0.7015	0.697

Tabla 1.1: Resultados en gráficas tipo A

5.3 Análisis y conclusiones sobre gráficas tipo A

De los resultados obtenidos de estos experimentos se obtienen varias observaciones interesantes sobre el desempeño de los algoritmos NORM y REF que podrían entrar en conflicto con las aseveraciones generales de Kumar et al. [11]. Los autores afirman en su artículo:

> *"En una configuración en la que no se apliquen cotas a la longitud de las trayectorias por omisión, el algoritmo refinado para bosques generadores (REF) crea árboles de transiciones por omisión de igual peso pero relativamente menor diámetro que el algoritmo normal (NORM)."*

así como,

> *"Cuando se aplican cotas sobre los diámetros, el algoritmo refinado genera bosques que además (de tener menor diámetro promedio) tienen mayor peso total."*

Los resultados de la Tabla 5.1 parecen corroborar la primera afirmación de los autores pues se cumple en todos los casos, para todos los órdenes se tiene que en la totalidad de las ejecuciones el peso de ambos bosques generadores fue igual, mientras tanto la distancia promedio

		Dist Máx×Omisión			Valencia Máx Prom			Aristas Promedio			% aristas usadas			% peso total		
Nodos	Cota	NORM	REF	BGS	NORM	REF	BGS	NORM	REF	BGS	NORM	REF	BGS	NORM	REF	BGS
30	∞	6.37	6.27	6.28	4.53	4.53	4.54	29	29	29	7.018%	7.018%	7.018%	25.678%	25.678%	25.678%
30	20	6.47	6.37	6.38	4.62	4.63	4.64	29	29	29	7.018%	7.018%	7.018%	25.835%	25.835%	25.835%
30	10	4.99	4.99	4.99	4.84	4.87	4.89	28.99	28.99	28.99	7.015%	7.015%	7.015%	25.533%	25.533%	25.529%
30	7	3.99	3.99	3.99	4.63	4.63	4.62	28.15	28.18	28.14	6.812%	6.819%	6.809%	24.906%	24.916%	24.900%
30	4	2	2	2	4.56	4.56	4.49	25.98	26.02	25.9	6.287%	6.296%	6.267%	22.876%	22.888%	22.843%
30	2	1	1	1	4.02	4.04	3.94	21.07	21.08	21	5.099%	5.101%	5.082%	18.655%	18.659%	18.634%
20	∞	5.16	5.1	5.1	4.34	4.36	4.36	19	19	19	10.526%	10.526%	10.526%	34.358%	34.358%	34.358%
20	10	4.77	4.75	4.75	4.3	4.33	4.36	19	19	19	10.526%	10.526%	10.526%	33.398%	33.403%	33.403%
20	7	3.97	3.96	3.95	4.63	4.66	4.64	18.83	18.84	18.82	10.432%	10.438%	10.427%	33.389%	33.407%	33.396%
20	4	2	2	2	4.51	4.54	4.52	17.43	17.44	17.41	9.657%	9.662%	9.645%	31.304%	31.319%	31.289%
20	2	1	1	1	3.78	3.81	3.71	14.02	14.03	13.95	7.767%	7.773%	7.729%	25.128%	25.142%	25.079%

Tabla 1.1: Resultados en gráficas tipo A

a la raíz como la distancia máxima por omisión siempre es mejor (menor) para el algoritmo refinado. Al explicar esto, se deben notar dos aspectos fundamentales del caso en que no se acota la longitud de las trayectorias por omisión, o lo que es igual, el diámetro de los árboles en el bosque generador. Primero, puesto que para estos experimentos se utilizan gráficas casi completamente conectadas[4], las probabilidades apuntan casi indiscutiblemente a que, en todos los casos, los bosques generadores resultantes de correr los algoritmos sin una cota se compondrán de una sola componente conexa, o lo que es lo mismo, serán un solo árbol. Esto se observa del hecho de que el número promedio de aristas en el bosque generador para cualquier tamaño de gráfica en la Tabla 5.1 es exactamente uno menos que el número de nodos, es decir, para cada una de las ejecuciones el resultado es un árbol generador. Dado esto, en segundo lugar se debe observar que para NORM, el algoritmo resultante al dejar de aplicar cotas es exactamente el algoritmo de Kruskal para árboles generadores máximos mientras que para REF es precisamente un algoritmo de Kruskal "refinado", pues en caso de empate en los pesos se agregará la arista que represente menor crecimiento para el diámetro. Pero el algoritmo de Kruskal entrega forzosamente alguno de los árboles de peso maximal,

[4] Decimos que son casi completamente conectadas pues en cada gráfica tipo A de n nodos, cada uno de los vértices tendrá un promedio de $0.95(n-1)$ vecinos.

en caso de haber mas de uno, o el de peso máximo en caso de ser único. Esto se puede garantizar por el hecho de que las aristas se van agregando de mayor a menor peso; como en cada ciclo en la gráfica se deberá descartar alguna arista, la arista eliminada será siempre la que se analice al último, o lo que es lo mismo, la de menor peso (o una de las de menor peso en caso de haber empate). Por lo tanto, sin importar si existe o no un criterio de desempate entre aristas del mismo peso, tanto REF como NORM entregarán un árbol de peso maximal. Resulta también bastante claro que en las ejecuciones sin acotar el algoritmo refinado también entregue árboles con menor diámetro que el algoritmo NORM pues en caso de empate en los pesos maximales intenta minimizar dicha variable, mientras que el primer algoritmo agrega aristas indiscriminadamente. Se sigue de ello que en cualquier gráfica que presente aristas de igual peso que se deseen agregar al árbol generador, la minimización del diámetro por parte de REF entregará un árbol generador de a lo más igual diámetro que NORM y en la mayoría de los casos con diámetro menor.

Ahora bien, la segunda afirmación citada arriba no parece coincidir completamente con los resultados de las ejecuciones. En general, se observan patrones de comportamiento bastante claros. Cuando se aplican cotas, el algoritmo REF entrega mejores resultados que NORM en todos los casos en relación al peso total del bosque generador. Sin embargo, cuando las cotas son relativamente pequeñas respecto al orden de las gráficas, NORM minimiza la distancia promedio a la raíz de mejor forma que REF. Esto se debe, sorprendentemente, al mejor funcionamiento del algoritmo REF pues en las ocasiones en las que la distancia promedio a la raíz es menor en el bosque generador de NORM, ocurre siempre que el bosque generador de REF contiene más aristas que el de NORM dando como resultado un mayor número de hojas. Al haber una relación hojas/árboles mayor en el bosque generador de REF, la distancia promedio a la raíz crece proporcionalmente. Sin embargo, si se retirarán del bosque generador de REF las últimas k aristas que se agregaron de modo que quedaran el mismo número de aristas que en el bosque generador de NORM, entonces se obtiene que el diámetro promedio a la raíz del primer bosque es a lo más aquel del segundo. Se concluye de esto que a pesar de que la segunda afirmación de los autores es incorrecta, pues no discierne

la relación entre el número de aristas en el bosque generador y la distancia promedio a la raíz, y por ende se encuentran escenarios en los que NORM minimiza de mejor forma este último valor. Independientemente de esta observación, la segunda afirmación es correcta si se toma en consideración la relación antes mencionada. Así, obtenemos que el desempeño del algoritmo REF es mejor al del algoritmo NORM en todos los casos al ejecutarse sobre gráficas de tipo A.

En cuanto al algoritmo BGS, los resultados de la Tabla 5.1 no parecen demasiado alentadores. En los casos sin acotar[5], la distancia promedio a la raíz es significativamente menor en BGS que en REF o en NORM, lo que significa una mejoría en el desempeño. Sin embargo, cuando se aplican cotas a las trayectorias por omisión, los resultados cambian drásticamente. Bajo estas cotas, el peso total de los bosques generadores de BGS parece ser menor incluso al del algoritmo NORM que agrega aristas indiscriminadamente. Se observa que nuevamente, el fenómeno se debe a que BGS agrega, en general, menos aristas que los demás algoritmos. Esto se explica por la tendencia que tiene BGS a generar nuevas mantener muchas componentes conexas independientes para reducir al mínimo el crecimiento general de los árboles en el bosque generador. La idea fundamental es que más adelante, estos árboles se puedan unir creando árboles más grandes que satisfagan las cotas y a su vez contengan una gran cantidad de aristas. Sin embargo, este comportamiento de 'optimización' juega en contra de BGS gracias a las características de las gráficas tipo A. En algunas ocasiones, la gran densidad de aristas aunado a la división de árboles de tamaños similares implican que el intentar unir dichos árboles generará violaciones a la cota sobre las trayectorias por omisión máximas. Es decir que la relación entre el orden de la gráfica de reducción y el tamaño de las cotas tiene una importancia fundamental en el desempeño de BGS sobre gráficas de tipo A. Si se compara el desempeño de de BGS con el de NORM, se observa que en los casos en que la cota es suficientemente grande, los árboles del algoritmo NORM crecen rápidamente y alcanzan la cota antes que los árboles de BGS, lo cual le impide agregar más aristas a los extremos de esos árboles más adelante, pues significaría una violación a la cota. En contraste,

[5] En estos escenarios nuevamente se tiene que BGS es un algoritmo de Kruskal 'refinado', por lo que el peso total del bosque generador es tan bueno como el de los demás algoritmos.

BGS, al mantener sus árboles con diámetros menores puede continuar agregando aristas más adelante, e incluso uniendo árboles de tamaños considerables. En cambio, cuando las cotas son pequeñas, el algoritmo BGS puede encontrarse con el caso en que al haber construido muchos árboles de diámetros reducidos, en iteraciones avanzadas ya no podrá agregar más aristas pues éstas unirían a los árboles que conectan en un árbol de dimensiones mayores a las permitidas, con lo cual muchos árboles terminan siendo demasiado chicos pero sin posibilidades de agregar más aristas. Bajo estas condiciones NORM parece comportarse mejor al agregar indiscriminadamente aristas y alcanzar rápidamente crecimientos máximos en algunos de sus árboles, pues hacia el final puede continuar agregando aristas que no signifiquen un aumento a los diámetros de esos árboles o bien aquellas que unan árboles que no hayan sido utilizados demasiado y que en consecuencia no hayan alcanzado la cota aún. Un caso extremo de este comportamiento se ejemplifica en la figura 5.1, en donde se muestran los árboles generadores resultantes de aplicar ambos algoritmos (NORM y BGS) en una gráfica completa de ocho vértices, K_8, con un valor de 2 para la cota a la longitud de las trayectorias por omisión en la que además los pesos de las aristas son todos iguales. Si suponemos que se analizan con el orden lexicográfico, es decir la arista (0, 1) seguida de la (0, 2), ... , seguida de la (0, 7), seguida de la (1, 2), etc., entonces NORM agregará aristas en el siguiente orden: (0, 1) , (0, 2) , (0, 3) , (0, 4) , (0, 5) , (0, 6) , (0, 7) pues dado el orden, son las primeras aristas que encuentra y ninguna de ellas ocasiona que se sobrepase la cota. Al haberse agregado 7 aristas, el programa termina (puesto que no se pueden agregar más de $n - 1$ aristas a una gráfica de n nodos sin crear un ciclo) dando como resultado el árbol generador de la figura 5.1a[6].

En cambio, siguiendo también un orden de análisis lexicográfico, BGS agregará aristas de la siguiente forma:

[6] Se observa que el algoritmo REF genera exactamente el mismo bosque generador que NORM para esta gráfica completa, pues también analiza las aristas en orden lexicográfico. La primera arista, (0, 1), supone un crecimiento de 1 en la trayectoria máxima del bosque. Las aristas restantes siguen representando un incremento unitario por lo que se agregar[a la primera que se encuentre, (0, 2). Después, todas las aristas restantes incidentes en el nodo 0 no representan crecimiento alguno, por lo que serán elegidas.

Primero agregará a (0, 1), luego evitará todas las aristas que incidan en los nodos 0 ó 1 pues ellas significarían un crecimiento del diámetro de ese árbol a 2, cuando aún hay aristas que ofrecen un crecimiento menor; así que se encuentra primero, dado el ordenamiento, con (2, 3). Siguiendo el mismo razonamiento, evita todas las aristas incidentes en los nodos 0, 1, 2, ó 3 y así agrega a la (4, 5) y repitiendo el proceso a la (6, 7). Después de esto, aunque el diámetro de los árboles se ha mantenido tan pequeño como ha sido posible, la cota es tan chica que cualquier nueva arista que se desee agregar unirá dos árboles por sus extremos (dado que solamente tienen extremos al ser dos vértices unidos por una arista) generando un nuevo árbol con diámetro 3 que transgredería la cota, por lo que no se pueden agregar más aristas y el resultado final se observa en la figura 5.1b.

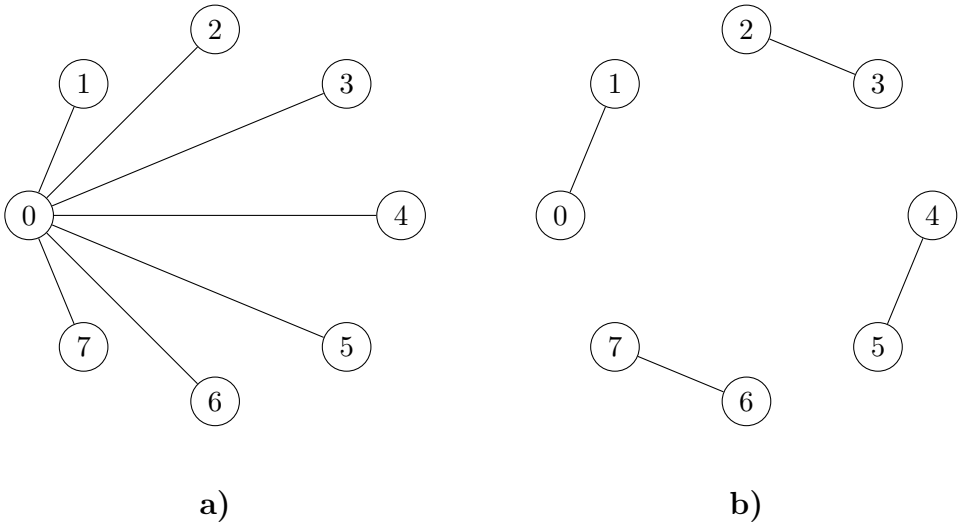

Figura 5.1: Bosques generadores para la gráfica K_8 obtenidos por los algorimos NORM (a) y BGS (b).

Otros valores llamativos en la Tabla 5.1 se observan en las últimas seis columnas, en las cuales se distingue que los pesos totales de los bosques generadores representan un porcentaje muy bajo del peso total de las gráficas de reducción para los tres algoritmos, mientras que el porcentaje del número total de aristas que se utiliza en los bosques generadores es aún menor. Este fenómeno se explica también a partir de las propiedades de las gráficas de reducción que se generaron para los experimentos. Para cada orden n, las gráficas de reducción elaboradas

por la aplicación asignan pesos aleatoriamente a todas las aristas de una gráfica completa (K_n) y sólo se eliminan de la gráfica completa aquellas aristas que reciban una asignación de peso igual a cero; dada la probabilidad de éxito de $\pi = 0.05$ en la distribución geométrica que se utiliza, esto significa que sólo se retiran el 5% de las aristas. En consecuencia, el número de aristas calculado en las gráficas de reducción de orden n es igual al número de aristas en K_n menos el número de aristas, fuera del total, a las que se les asigna peso cero. Si llamamos $|A_n|$ a dicho número de aristas, tenemos entonces:

$$|A_n| = \frac{n(n-1)}{2} - \pi\frac{n(n-1)}{2} = 0.95\frac{n(n-1)}{2}$$

Recordamos de la teoría de gráficas que el número de aristas en un bosque generador es igual a $n - m$, en donde n denota el número de vértices en la gráfica original mientras que m denota el número de componentes conexas en el bosque generador. Es decir que si lo que se quiere es maximizar el número de aristas en un bosque generador, en el mejor de los casos éste se compondrá de una única componente conexa y, por tanto, contendrá $n - 1$ aristas. En las ejecuciones que dan forma a los resultados de la Tabla 5.1 sólo puede acontecer este mejor caso en algunos escenarios, aquellos en los que no se aplica una cota para la longitud de las trayectorias por omisión y aquellos en los que la cota es lo suficientemente grande para evitar la necesidad de dividir el bosque generador en más de una componente conexa. En todos los demás casos, el número de componentes conexas en el bosque generador es mayor a uno, reduciendo así el número de aristas utilizadas, cifra que empeora a medida que la cota a los diámetros se reduce, pues los árboles son restringidos a un menor crecimiento cada vez. Si nos enfocamos en el mejor de los casos, se tiene que se utilizan $n - 1$ aristas de las $|A_n|$ que contiene la gráfica de reducción, obteniéndo la siguiente relación:

$$\frac{n-1}{|A_n|} = \frac{1}{0.475n}$$

El resultado es muy pequeño incluso desde los órdenes menores del experimento, además de que es inversamente proporcional al crecimiento de n, por lo que para órdenes mayores el porcentaje de aristas utilizadas es cada vez menor. Estos mismos razonamientos explican el bajo porcentaje del peso total del bosque generador en relación a la gráfica de reducción, pues

aunque se intenta agregar las aristas de mayor peso siempre que se pueda, el bajo porcentaje de aristas que se puede utilizar limita en gran medida el crecimiento de este valor. Si se toma como ejemplo el promedio de las ejecuciones sobre 100 nodos sin acotar, se tiene que aunque el porcentaje de aristas utilizadas es apenas mayor al 2%, el porcentaje del peso total, aunque bajo, es una cifra mucho mayor cercana al 10.3%. Esto se debe a que la elección de aristas se hace de mayor a menor por lo que con todo y que el peso medio estadístico de las aristas es de 19, el promedio del peso en las aristas elegidas para estos bosques generadores es de 93.56, de ahí la diferencia entre ambos porcentajes.

5.4 Ejecuciones sobre gráficas de reducción tipo B

El análisis desarrollado con los resultados de las ejecuciones sobre gráficas de tipo A muestra que el desempeño de los algoritmos varía considerablemente debido a las diferentes características de las gráficas sobre las que se ejecutan. Se nota así la importancia escencial de realizar algunos experimentos sobre gráficas de tipo B que se asemejan en gran medida a las gráficas de reducción provenientes de autómatas similares a aquellos utilizados en las aplicaciones informáticas reales. De esta forma, se podrán obtener datos válidos sobre el desempeño de los diferentes algoritmos en aplicaciones prácticas. Para ello, se corrieron los algoritmos bajo cuatro escenarios en gráficas tipo B de distintos órdenes de magnitud, consistiendo de 1000, 2000 y 4000 vértices. En dos escenarios no se acota la longitud de las trayectorias por omisión, mientras que en los otros dos se contempla una cota donde el diámetro máximo aceptable tenga longitud 4. Además, entre cada uno de estos pares de escenarios se distinguen variaciones en el valor de la probabilidad de éxito en la función de distribución geométrica (π). Los valores para π son 0.9 y $\frac{1}{3}$. Los resultados obtenidos al correr los algoritmos sobre estas gráficas se resumen en las Tablas 5.2 a la 5.5.

Sin Cotas $\pi = 0.9$	Nodos		1000	2000	4000
	Cota		∞	∞	∞
	Iteraciones		100	100	100
	NORM mejor		0	0	0
	REF mejor		0	0	0
	BGS mejor		0	0	0
	Iguales		100	100	100
	Transiciones Totales		255702	511404	1022809
	Duplicados		250000	500000	1000000
	Aristas Totales		1177	2353	4706
	Ciclos		178	354	707
NORM (bosque generador)	Peso Total		249395,47	499040,58	998330,80
	Dist. Prom. a Raíz		16,711	22,149	30,402
	Dist. Max. por Omisión		32,6	43,5	61
	Aristas		999	1999	3999
REF (bosque generador)	Peso Total		249395,47	499040,58	998330,80
	Dist. Prom. a Raíz		8,862	9,588	10,818
	Dist. Max. por Omisión		16,2	17,9	19,6
	Aristas		999	1999	3999
BGS (bosque generador)	Peso Total		249395,47	499040,58	998330,80
	Dist. Prom. a Raíz		12,861	14,622	18,436
	Dist. Max. por Omisión		21,9	26,5	32
	Aristas		999	1999	3999
AFD^2 de NORM	Transiciones Totales		6306,69	12363,74	24477,83
	% Reducción		97,53%	97,58%	97,61%
AFD^2 de REF	Transiciones Totales		6306,69	12363,74	24477,83
	% Reducción		97,53%	97,58%	97,61%
AFD^2 de BGS	Transiciones Totales		6306,69	12363,74	24477,83
	% Reducción		97,53%	97,58%	97,61%

Tabla 5.2: Resultados para gráficas tipo B, sin cotas y $\pi = 0.9$.

5.5 Análisis y conclusiones sobre gráficas tipo B

Es posible observar en los resultados, desplegados en las tablas 5.2 a 5.5 (arriba y en las páginas subsiguientes), que sin importar los cambios en la distribución de pesos (π) y en la magnitud de las gráficas de reducción, la minimización de los valores que definirán la complejidad en el número de accesos a memoria, como son la distancia promedio a la raíz y la distancia máxima por omisión, continúan siendo al menos tan buenos para el algoritmos BGS como lo son para el algoritmo NORM. En las ejecuciones donde no se aplica una cota a los diámetros (Tablas 5.2 y 5.3), los valores de BGS son ampliamente mejores que los de NORM pues, promediando los resultados, se observa que la distancia promedio a la raíz de los vértices en los bosques generadores de NORM es 50% mayor que en los de BGS cuando

Sin Cotas $\pi = \frac{1}{3}$	Nodos		1000	2000	4000
	Cota		∞	∞	∞
	Iteraciones		100	100	100
	NORM mejor		0	0	0
	REF mejor		0	0	0
	BGS mejor		0	0	0
	Iguales		100	100	100
	Transiciones Totales		255702	511404	1022809
	Duplicados		250000	500000	1000000
	Aristas Totales		1186	2372	4743
	Ciclos		187	373	744
NORM (bosque generador)	Peso Total		248011,85	496289,00	992743,55
	Dist. Prom. a Raíz		18,778	26,407	31,044
	Dist. Max. por Omisión		38,5	51,8	63,1
	Aristas		999	1999	3999
REF (bosque generador)	Peso Total		248011,85	496289,00	992743,55
	Dist. Prom. a Raíz		15,550	19,194	23,660
	Dist. Max. por Omisión		28,5	36,1	43,9
	Aristas		999	1999	3999
BGS (bosque generador)	Peso Total		248011,85	496289,00	992743,55
	Dist. Prom. a Raíz		16,675	19,871	24,308
	Dist. Max. por Omisión		32	38,8	48,3
	Aristas		999	1999	3999
AFD^2 de NORM	Transiciones Totales		7690,31	15115,32	30065,08
	% Reducción		96,99%	97,04%	97,06%
AFD^2 de REF	Transiciones Totales		7690,31	15115,32	30065,08
	% Reducción		96,99%	97,04%	97,06%
AFD^2 de BGS	Transiciones Totales		7690,31	15115,32	30065,08
	% Reducción		96,99%	97,04%	97,06%

Tabla 5.3: Resultados para gráficas tipo B, sin cotas y $\pi = \frac{1}{3}$.

$\pi = 0.9$ y 25.8% mayor cuando $\pi = \frac{1}{3}$. Similarmente, el valor de la distancia por omisión máxima resulta, también para NORM, 70.5% mayor con $\pi = 0.9$ y superior en 29% para $\pi = \frac{1}{3}$. De todo esto se sigue que, sin importar la distribución de los pesos en las aristas de las gráficas de reducción que se puedan obtener, el desempeño de la estructura AFD^2 generada por BGS es entre 20% y 80% más veloz que la estructura generada por NORM al consumir símbolos de la entrada de datos cuando no se asignan cotas a las trayectorias por omisión que dichas estructuras puedan contener. Al comparar con el algoritmo REF observamos que, en lo que se refiere a trayectorias máximas por omisión y distancia promedio a la raíz, los resultados de los bosques generadores producidos por este útlimo algoritmo son inclusive mejores que los obtenidos por BGS. Cuando la distribución de los pesos en la

Cota = 4 π **= 0.9**	Nodos	1000	2000	4000
	Cota	4	4	4
	Iteraciones	100	100	100
	NORM mejor	0	0	0
	REF mejor	4	0	0
	BGS mejor	96	100	100
	Iguales	0	0	0
	Transiciones Totales	255702	511404	1022809
	Duplicados	250000	500000	1000000
	Aristas Totales	1177	2353	4706
	Ciclos	178	354	707
NORM (bosque generador)	Peso Total	232171,43	466528,57	931660,71
	Dist. Prom. a Raíz	1,111	1,121	1,122
	Dist. Max. por Omisión	2	2	2
	Aristas	764,8	1536,8	3069
REF (bosque generador)	Peso Total	233507,14	466862,50	932601,79
	Dist. Prom. a Raíz	1,115	1,118	1,117
	Dist. Max. por Omisión	2	2	2
	Aristas	769,2	1537,9	3072,1
BGS (bosque generador)	Peso Total	240337,50	480492,86	959255,36
	Dist. Prom. a Raíz	1,062	1,066	1,063
	Dist. Max. por Omisión	2	2	2
	Aristas	791,7	1582,8	3159,9
AFD^2 de NORM	Transiciones Totales	23530,73	44875,74	91147,92
	% Reducción	90,80%	91,22%	91,09%
AFD^2 de REF	Transiciones Totales	22195,02	44541,82	90206,85
	% Reducción	91,32%	91,29%	91,18%
AFD^2 de BGS	Transiciones Totales	15364,66	30911,46	63553,28
	% Reducción	93,99%	93,96%	93,79%

Tabla 5.4: Resultados para gráficas tipo B, cota = 4 y $\pi = 0.9$.

gráfica de reducción esta, en general, en un rango corto, se obtiene que el valor para la distancia promedio a la raíz en los bosques de BGS es 56.9% mayor a la cifra de REF y la distancia máxima por omisión es, en promedio, 49.7% más larga. Sin embargo cuando se distribuyen mejor los pesos de las aristas (i.e. cuando $\pi = \frac{1}{3}$), las mejoras de REF son ya solamente del 4% y 9% respectivamente. Se puede concluir sin lugar a dudas que los AFD^2 generados por REF cuando no se acotan las trayectorias por omisión obtendrán el mejor desempeño en lo que a velocidad de ejecución se refiere. No obstante, a medida que los pesos en las gráficas de reducción se distribuyen de forma más homogénea[7], el rendimiento

[7] La distribución de pesos en las gráficas de reducción depende ampliamente de las características que tengan las expresiones regulares que se utilicen para construir el autómata finito determinista. En conjuntos de expresiones regulares que contengan solamente la cerradura de Kleene y la concatenación como operadores,

Cota = 4 $\pi = \frac{1}{3}$	Nodos		1000	2000	4000
	Cota		4	4	4
	Iteraciones		100	100	100
	NORM mejor		0	0	0
	REF mejor		47	1	0
	BGS mejor		53	99	100
	Iguales		0	0	0
	Transiciones Totales		255702	511404	1022809
	Duplicados		250000	500000	1000000
	Aristas Totales		1186	2372	4743
	Ciclos		187	373	744
NORM (bosque generador)	Peso Total		235835,60	470218,69	939875,83
	Dist. Prom. a Raíz		1,115	1,111	1,112
	Dist. Max. por Omisión		2	2	2
	Aristas		780,5	1556,3	3110,9
REF (bosque generador)	Peso Total		237030,00	472642,38	943754,17
	Dist. Prom. a Raíz		1,116	1,116	1,114
	Dist. Max. por Omisión		2	2	2
	Aristas		784,5	1564,4	3123,8
BGS (bosque generador)	Peso Total		237213,33	476068,33	951280,24
	Dist. Prom. a Raíz		1,106	1,111	1,112
	Dist. Max. por Omisión		2	2	2
	Aristas		785	1575,7	3148,7
AFD^2 de NORM	Transiciones Totales		19866,56	41185,63	82932,80
	% Reducción		92,23%	91,95%	91,89%
AFD^2 de REF	Transiciones Totales		18672,16	38761,94	79054,47
	% Reducción		92,70%	92,42%	92,27%
AFD^2 de BGS	Transiciones Totales		18488,82	35335,98	71528,39
	% Reducción		92,77%	93,09%	93,01%

Tabla 5.5: Resultados para gráficas tipo B, cota= 4 y $\pi = \frac{1}{3}$.

de las estructuras generadas por BGS es prácticamente tan bueno como el de REF. En estas mismas ejecuciones vemos que, al no acotarse la longitud de los diámetros, tenemos nuevamente variaciones del algoritmo de Kruskal para obtener árboles generadores, por lo que se espera, y de hecho así sucede, que los pesos totales de los bosques generados por los tres algoritmos sean idénticos. De mucho mayor interés resulta el hecho de que, considerando las propiedades de las gráficas de reducción, ahora se logra una reducción muy significativa del número total de transiciones entre el AFD original y el AFD^2. Para los casos en que no se aplican cotas a la longitud de las trayectorias por omisión se obtiene que cualquiera de los

se obtendrá una distribución con menor rango. En AFD construidos por un conjunto de expresiones regulares que también utilicen la unión como operador se obtendrán, generalmente, gráficas de reducción con pesos distribuidos en rangos de mucha mayor amplitud.

AFD^2 resultantes entregan una compactación superior al 97.5% en el número de transiciones con respecto al AFD original. Esto sucede para cualquiera de las magnitudes y sin importar cuál algoritmo se utilice para generar la estructura. En cuanto a las ejecuciones para las que se especificó una cota sobre la longitud de los diámetros (Tablas 5.4 y 5.5), el reducido valor de la cota garantiza un desempeño muy alto en cuanto a tiempos de ejecución para cualquiera de los AFD^2 que se generan, pues en el peor caso se recorrerán dos transiciones por omisión antes de consumir un símbolo de la entrada. El valor de 4 para la cota resulta tan reducida que limita en gran medida el crecimiento de los valores relacionados a las distancias sobre el bosque generador. Difícilmente se generará una gráfica de reducción que pueda evitar que el crecimiento de al menos uno de los árboles en el bosque alcance la cota, por lo que la distancia por omisión máxima será de 2 en todas la ejecuciones de todos los algoritmos. Por lo mismo no se logran observar las grandes diferencias de los escenarios sin acotar. Sin embargo, la distancia promedio a la raíz sí refleja diferencias en la minimización alcanzada por los distintos algoritmos. El cambio más significativo sobre los comparativos de este valor proviene de notar que los AFD^2 construidos por el algoritmo BGS se vuelven ahora en las estructuras con el mejor desempeño en este renglón, ya que además de ser menor en todos los casos, promedia una mejoría del 5%. No obstante la mejoría en cuanto al desempeño en los tiempos de ejecución, el resultado notable en estos escenarios surge del hecho de que el algoritmo BGS presenta pesos totales muy superiores a los de NORM y REF en sus bosques generadores, incluso con cotas tan pequeñas como 4 en relación a las magnitudes de las gráficas de hasta 4000 vértices. Es decir que para todas las gráficas de reducción, el algoritmo simplificado propuesto en este trabajo, no sólo produjo bosques generadores de mayor peso que los otros algoritmos, sino que también con menor distancia promedio a la raíz. A pesar de que las reducciones entre el número de transiciones del AFD original y el AFD^2 no son tan sobresalientes como en el escenario sin cota, siguen siendo muy importantes. El algoritmo básico entrega una compactación promedio cercana al 92% cuando $pi = \frac{1}{3}$ y de 91.1% con $pi = 0.9$, a su vez, REF alcanza reducciones promedio del 92.5% y del 91.3% respectivamente. Finalmente, el nuevo algoritmo BGS obtiene los mejores resultados

al compactar en un 92.96% con el primer escenario y del 93.91% en el segundo. Si se toman como ejemplo los promedios de las cien ejecuciones con $\pi = 0.9$ y con cota de 4 sobre la longitud de las trayectorias por omisión en autómatas con 2000 estados, se observa que de las 511,404 transiciones que posee el autómata original, el AFD^2 generado por el algoritmo BGS contiene solamente 30,912 transiciones. Si se compara esta última cifra con la cantidad de transiciones que contienen las estructuras generadas por NORM y REF, de 44,876 y 44,541 transiciones respectivamente, se observa que la mejoría del algoritmo BGS en términos de la memoria necesaria para alojar las estructuras es muy significativa. En este caso se requiere de un 45.17% más memoria para alojar la estructura de NORM de la que se necesita para la de BGS. En relación a REF, el AFD^2 que genera necesita un 44.09% más memoria que el del nuevo algoritmo. En promedio sobre todos los escenarios acotados, se las estructuras que se obtienen con los algoritmos NORM y REF solicitan 29.07% y 24.77% respectivamente más memoria que los AFD^2 generador por BGS.

Cuando se piensa que la motivación principal para generar algoritmos diferentes al de Joseph Kruskal fue la de poder acotar los tiempos de ejecución a valores que entreguen un desempeño sobresaliente al analizar la entrada de datos en las redes de compudoras incluso en el peor caso, se comprende que los escenarios sin acotar no resultan viables para aplicaciones reales y sencillamente nos permiten entender a fondo las cualidades y limitaciones de cada algoritmo. En la práctica, se buscará acotar al máximo la distancia promedio a la raíz en los bosques generadores de las gráficas de reducción pues esa será una medida importante del desempeño en tiempo de ejecución del análisis profundo de paquetes con los AFD^2. Este último razonamiento subraya la importancia de las mejoras en el desempeño por parte del algoritmo BGS pues pueden representar la obtención de resultados superiores a los reportados en [11] en aplicaciones prácticas, los cuales de por sí, son sorprendentes. Otro factor prometedor del algoritmo BGS proviene de la velocidad con que genera los AFD^2 pues al simplificar el criterio de desempate al elegir entre aristas válidas del mismo peso, se ejecuta en mucho menor tiempo que el algoritmo REF. Como los algoritmos estudiados en este trabajo solamente se ejecutan para recalcular el AFD^2 cada que se modifica el conjunto

de expresiones regulares que sirven como patrones para el apareamiento, las velocidades con las que se ejecuten parecen carecer de importancia. Sin embargo, en escenarios donde el conjunto de patrones se modifique regularmente o en aplicaciones cuya base de datos contenga un número muy grande de expresiones regulares, esta diferencia en los tiempos de ejcución podría también ser un factor relevante para recurrir a BGS por sobre los demás algoritmos.

5.6 El algoritmo MOV

En esta sección se describe sucintamente un último algoritmo, el de maximización orientada a valencias (MOV), junto con sus resultados y un breve análisis sobre éstos. Este último algoritmo no presento mejorías de ningún tipo sobre el desempeño de los otros tres, además de ser más complicado de codificar y mucho más lento al ejecutarse para generar un AFD^2 inclusive que el algoritmo REF. El algoritmo surge de una observación sobre un conjunto de resultados preliminares que comparaban el desempeño de los algoritmos NORM y REF. En dichos resultados se observó un comportamiento relativamente común, aunque de ninguna manera constante. Los resultados del peso total en los bosques generadores aparentaban favorecer al algoritmo que tuviera un valor mayor en el rubro de las valencias máximas de cada iteración. A pesar de que este comportamiento no era constante, parecía ocurrir en la gran mayoría de los casos cuando las diferencias en este valor eran lo suficientemente amplias. Bajo esta motivación se diseño e implementó el algoritmo MOV. La problemática esecial en su codificación surge de la forma en que agrega aristas pues en lugar de agregarlas una a la vez, MOV debe ser capaz de agregar un conjunto de aristas adyacentes al mismo vértice. Esto obliga a utilizar varias estructuras temporales y a utilizar un mayor número de ciclos que en los demás algoritmos disminuyendo su desempeño en tal medida que pudiera resultar inviable para generar los AFD^2 a partir de autómatas con más de 50,000 estados. Una vez codificado, se realizaron pruebas comparativas junto con los algoritmos NORM, REF y BGS sobre gráficas tipo A y B. Los resultados se resumen en las tablas 5.6 y 5.7.

En la tabla 5.6, para gráficas tipo A, se observa que los resultados entre los algoritmos NORM, REF y BGS se comportan similarmente a los obtenidos para la tabla 5.1. En

PI = 1/20	Nodos	100	100	100	100	200	200	200	200	600	600	600	600
	Cota	∞	10	4	2	∞	10	4	2	∞	10	4	2
	Iter.	30	30	30	30	30	30	30	30	30	30	30	30
	NORM	0	0	0	0	0	0	0	0	0	1	0	0
	REF	0	6	8	7	0	4	8	8	0	21	22	17
	BGS	0	2	2	0	0	2	4	1	0	3	3	1
	MOV	0	0	0	0	0	0	3	0	0	3	1	2
	Iguales	30	15	16	12	30	9	6	10	30	0	0	0
	Ciclo	0	7	4	11	0	15	9	11	0	2	4	10
Peso Total	NORM	9189,8	8996,4	8023,2	6592,2	21312,0	20660,3	18571,8	15047,8	76875,2	74750,2	66773,4	54359,6
	REF	9189,8	9001,7	8044,5	6603,1	21312,0	20671,4	18587,4	15060,0	76875,2	74811,9	66865,2	54432,0
	BGS	9189,8	8995,1	8023,6	6572,5	21312,0	20660,5	18567,2	15008,5	76875,2	74728,7	66721,6	54179,4
	MOV	9189,8	8996,1	8024,3	6592,0	21312,0	20656,3	18578,1	15047,8	76875,2	74763,2	66776,9	54366,3
Distancia Prom a la Raíz	NORM	6,511	3,100	1,323	0,707	8,997	3,062	1,336	0,705	13,153	3,100	1,333	0,707
	REF	6,354	3,124	1,332	0,709	8,635	3,078	1,339	0,706	12,818	3,111	1,335	0,709
	BGS	6,333	3,094	1,322	0,703	8,554	3,055	1,331	0,702	12,608	3,073	1,328	0,704
	MOV	6,503	3,098	1,324	0,707	8,905	3,063	1,337	0,705	13,158	3,108	1,333	0,707
Distancia Max por Omisión	NORM	12,100	5,000	2,000	1,000	17,067	5,000	2,000	1,000	26,200	5,000	2,000	1,000
	REF	11,733	5,000	2,000	1,000	16,467	5,000	2,000	1,000	25,533	5,000	2,000	1,000
	BGS	11,667	5,000	2,000	1,000	16,467	5,000	2,000	1,000	25,467	5,000	2,000	1,000
	MOV	12,033	5,000	2,000	1,000	16,933	5,000	2,000	1,000	26,100	5,000	2,000	1,000
Valencia Máxima	NORM	5,467	5,700	5,267	5,033	6,267	6,300	6,033	5,567	7,133	6,933	6,800	6,067
	REF	5,467	5,700	5,333	5,033	6,333	6,267	6,133	5,600	7,133	6,933	6,767	6,067
	BGS	5,500	5,667	5,300	5,033	6,400	6,200	6,100	5,533	7,133	6,900	6,733	6,000
	MOV	5,467	5,667	5,267	5,067	6,300	6,300	6,067	5,567	7,133	6,967	6,767	6,067
Número de Aristas	NORM	99,000	97,033	86,833	70,700	199,000	193,767	174,167	141,000	599,000	582,267	522,967	424,367
	REF	99,000	97,167	87,167	70,900	199,000	193,900	174,467	141,167	599,000	582,867	523,867	425,100
	BGS	99,000	97,000	86,833	70,300	199,000	193,700	174,033	140,433	599,000	581,867	522,300	422,400
	MOV	99,000	97,033	86,833	70,700	199,000	193,733	174,267	141,000	599,000	582,367	523,033	424,433

Tabla 5.6: Comparativo de MOV para gráficas tipo A.

cuanto al desempeño del algoritmo MOV, es fácil observar que la compactación realizada por REF es mejor para todas las combinaciones magnitud/cota analizadas. De hecho, en varios escenarios incluso el algoritmo el más simple, NORM, entrega mejores resultados. Además, en relación a las distancias por omisión tampoco ofrece mejoras significativas respecto al desempeño de los demás algoritmos e incluso en este rubro, sus resultados se asemejan a los del algoritmo NORM. Es interesante notar que el algoritmo ni siquiera logra cumplir con su meta fundamental de maximizar las valencia máxima de los bosques generadores pues en dicho rubro su desempeño es también muy similar al de los demás algoritmos, superado en varias ocasiones por cualquier otro que se elija. Esto es fácil de explicar en este caso pues la distribución geométrica obliga al algoritmo a agregar las aristas de mayor peso primero, las cuales se encuentran muy dispersas y en pequeñas cantidades, de forma que cuando se alcanzan los conjuntos de pesos que contienen múltiples aristas, la gran mayoría de ellas ya no son elegibles para ser agregadas pues las probabilidades de que incumplan alguno de los

criterios para poderse utilizar son bastante altas. En cambio si el algoritmo MOV se ejecuta sobre gráficas de reducción en las que a toda arista se le asigne el mismo peso, el algoritmo entregará pesos muy elevados y valencias máximas insuperables.

En cuanto a las ejecuciones sobre gráficas tipo B resumidas en la tabla 5.7, MOV se distingue como el peor algoritmo pues aunque sus distancias por omisión son ligeramente menores a las de NORM, esto se debe al hecho de que consigue agregar una cantidad menor de aristas, lo que a su vez le confiere un peso total significativamente menor. Salta a la vista la supremacía, sobre MOV, de los algoritmos REF y, en mayor medida, BGS con desempeños ampliamente superiores. Se observa también que en los dos casos acotados, MOV efectivamente consigue los valores más elevados en el rubro de valencias máximas, pero esto no se refleja ni remotamente en una mejoría al desempeño. Esta última observación nos lleva a confirmar que el criterio mismo que dió lugar a este algoritmo no es suficiente para diseñar una heurística que mejore el desempeño de las estudiadas arriba.

	Nodos	400	400	1000	1000
	Cota	∞	4	∞	4
PI = 1/3	Iteraciones	30	30	30	30
	NORM mejor	0	1	0	0
	REF mejor	0	2	0	1
	BGS mejor	0	26	0	29
	MOV mejor	0	0	0	0
	Iguales	30	0	30	0
	Ciclo	0	1	0	0
NORM	Peso Total	99479,73	93691,15	249078,04	235251,35
(bosque generador)	DistPromARaiz	12,46	1,11	18,78	1,11
	DistDefMáaxima	24,30	2,00	37,37	2,00
	Valencia Máxima	6,90	5,80	7,07	6,33
	Aristas	399,00	308,97	999,00	775,77
REF	Peso Total	99479,73	94073,57	249078,04	236070,00
(bosque generador)	DistPromARaiz	10,36	1,11	14,82	1,11
	DistDefMáxima	19,30	2,00	28,03	2,00
	Valencia Máxima	6,87	5,90	7,07	6,50
	Aristas	399,00	310,23	999,00	778,47
BGS	Peso Total	99479,73	95955,28	249078,04	239682,02
(bosque generador)	DistPromARaiz	10,41	1,10	14,72	1,10
	DistDefMáaxima	20,13	2,00	28,77	2,00
	Valencia Máxima	6,90	5,53	7,17	6,07
	Aristas	399,00	316,43	999,00	790,37
MOV	Peso Total	99479,73	93053,06	249078,04	231914,44
(bosque generador)	DistPromARaiz	11,54	1,14	17,64	1,14
	DistDefMáxima	23,00	2,00	35,57	2,00
	Valencia Máxima	6,87	6,43	7,23	7,00
	Aristas	399,00	306,87	999,00	764,77

Tabla 5.7: Resultados comparativos del algoritmo MOV sobre gráficas tipo B.

Capítulo 6

Conclusiones y perspectivas

Con base en un profundo análisis de los resultados obtenidos para los algoritmos NORM y REF, los cuales demostraron la absoluta superioridad del segundo en aplicaciones reales, se lograron desarrollar dos nuevos algoritmos para la generación de estructuras AFD^2 a partir de autómatas finitos. Estos nuevos algoritmos, fundamentalmente distintos en cuanto a su diseño y conceptualmente ajenos, entregaron resultados tan contrastantes como los propios algoritmos. El algoritmo MOV, diseñado a partir de observaciones sobre el desempeño de los dos algoritmos iniciales, que parecían indicar que aumentar las valencias de los nodos centrales en las componentes de los bosques generadores podría entregar un aumento en el desempeño al permitir que se agregara un número mayor de aristas, resultó un auténtico descalabro. No sólo fue incapaz de mejorar el desempeño del algoritmo más simple, sino que, para gráficas complejas, demostró que ni siquiera representa un procedimiento viable para aumentar significativamente las valencias máximas de los bosques generadores.

Simultáneamente se planteó otro algoritmo con base en más observaciones, el cual buscaba mejorar el desempeño del algoritmo REF, superior en los experimentos, variando un poco su comportamiento. Esta ligera variación conceptual no sólo entregó un algoritmo que se ejecuta en mucho menor tiempo, sino que además ofrece un rendimiento significativamente superior en autómatas con características parecidas a los que se pueden obtener de un conjunto real de expresiones regulares para detectar intrusiones en redes informáticas.

La obtención de este segundo algoritmo, bautizado como BGS, significa una aportación al campo del apareamiento de patrones basado en expresiones regulares por parte de este trabajo. Además, plantea nuevas opciones de experimentación a futuro para intentar proveer algoritmos con características muy superiores incluso a aquellos utilizados actualmente en la industria.

Sin lugar a dudas, los resultados obtenidos en el presente trabajo parecen contundentes. Esto significa una enorme motivación para continuar por el mismo trayecto en busca de algoritmos que representen mejorías aún mayores al desempeño, así como para la exploración de nuevas aplicaciones para estas estructuras.

Bibliografía

[1] A. Aho y M. Corasick, "Efficient string matching: An aid to bibliographic search", Communication of the ACM, 18(6):333-340, 1975.

[2] A. Amir, et. al., "Dynamic Text and Static Pattern Matching", ACM Transactions on Algorithms, Vol. 3, Num. 2, Article 19, Mayo 2007.

[3] S. Antonatos, K. Anagnostakis y E. Markatos, "Generating realistic workloads for network intrusion detection systems", ACM SIGSOFT Softw. Eng. Notes, 29(1):207–215, 2004.

[4] T. Cormen, et. al., "Introduction to Algorithms", The MIT Press y McGraw-Hill (2da edición), 2001.

[5] E. Dijkstra, "A Discipline of Programming", Prentice-Hall Series in Automatic Computation, 1976.

[6] M.Fisk y G. Varghese, "Applying Fast String Matching to Intrusion Detection", Technical Report CS2001-0670, UCSD, 2001.

[7] M. Goodrich y R. Tamassia, "Data Structures and Algorithms in Java", John Wiley & Sons, Inc. (4ta edición), 2006.

[8] D. Gusfield, "Algorithms on Strings, Trees and Sequences: Computer Science and Computational Biology", Cambridge University Press, 1997.

[9] J. Hopcroft, et.al., "Introduction to Automata Theory, Languages, and Computation", Addison Wesley (2da edición), 2000.

[10] J. Kruskal, "On the Shortest Spanning Subtree of a Graph and the Traveling Salesman Problem", Proceedings of the American Mathematical Society, 7(1):48–50, 1956.

[11] S. Kumar, et. al., "Algorithms to Accelerate Multiple Regular Expressions Matching for Deep Packet Inspection", ACM SIGCOMM, pp. 339–350, 2006.

[12] G. Navarro, y M. Raffinot, "Compact DFA Representation for Fast Regular Expression Search", Lecture Notes in Computer Science, 2141:1–12, 2001.

[13] G. Navarro, y M. Raffinot, "Flexible Pattern Matching in Strings", Cambridge University Press, 2002.

[14] L. Salmela, et. al., "Multipattern String Matching with q-Grams", ACM Journal of Experimental Algorithmics, 11(1.1):1–19, 2006.

[15] R. Sommer y V. Paxson, "Enhancing Byte-Level Network Intrusion Detection Signatures with Context", Proceedings ACM CCS, 2003.

[16] E. Viso, "Introducción a la teoría de la computación (Autómatas y lenguajes formales)", Temas de Computación, Las Prensas de Ciencias, 2008.

[17] B. Watson y R. Watson, "A new family of string pattern matching algorithms", South African Computer Journal, 30:34–41, 2004.

[18] S. Wu y U. Manber, "A fast algorithm for multi-pattern searching", Technical Report TR-94-17, Dept. of Computer Science, Univ. of Arizona, 1994.

[19] Fang Yu, et. al., "Fast and Memory-Efficient Regular Expression Matching for Deep Packet Inspection", ACM ANCS, pp. 93–102, 2006.