TESIS DEFENDIDA POR

JUAN PAULO ALVARADO MAGAÑA

Y aprobada por el siguiente comité:

Dr. José Alberto Fernández Zepeda

Director del Comité

Dr. Carlos Alberto Brizuela Rodríguez

Miembro del Comité

Dr. Luis Alejandro Márquez Martínez

Miembro del Comité

Dr. Pedro Gilberto López Mariscal

Coordinador del Programa de Ciencias

Dr. Raúl Ramón Castro Escamilla

Director de Estudios de Posgrado en Ciencias de la Computación

25 de Agosto de 2006

CENTRO DE INVESTIGACIÓN CIENTÍFICA Y DE EDUCACIÓN
SUPERIOR DE ENSENADA

PROGRAMA DE POSGRADO EN CIENCIAS EN
CIENCIAS DE LA COMPUTACIÓN

ANÁLISIS DE ALGORITMOS AUTO-ESTABILIZANTES PARA ELECCIÓN DE LÍDER

TESIS

que para cubrir parcialmente los requisitos necesarios para obtener el grado de

MAESTRO EN CIENCIAS

Presenta:

JUAN PAULO ALVARADO MAGAÑA

Ensenada, Baja California, México. Agosto de 2006.

RESUMEN de la tesis de **Juan Paulo Alvarado Magaña**, presentada como requisito parcial para obtener el grado de MAESTRO EN CIENCIAS en CIENCIAS DE LA COMPUTACIÓN. Ensenada, Baja California, México. Agosto de 2006.

ANÁLISIS DE ALGORITMOS AUTO-ESTABILIZANTES PARA ELECCIÓN DE LÍDER

Resumen aprobado por:

Dr. José Alberto Fernández Zepeda
Director de Tesis

La auto-estabilización es un paradigma de tolerancia a fallas, en donde un sistema distribuido puede llegar a un estado global válido a partir de cualquier estado arbitrario inválido en un número finito de pasos.

El tema de investigación de esta tesis son los algoritmos auto-estabilizantes para elección de líder en un árbol. Eligir un líder en cualquier grafo es asignar atributos particulares y únicos a un solo nodo del grafo. Un sistema distribuido puede perder su líder cuando sufre alguna perturbación, es obligación del algoritmo auto-estabilizante detectar el error y enseguida ejecutarse para elegir un líder nuevo.

En este trabajo de tesis se estudia en particular el algoritmo auto-estabilizante para elección de líder de Xu y Srimani (Xu y Srimani, 2005). Este algoritmo elige un líder en $O(n^4)$ pasos (donde n es el número total de nodos en el árbol) utilizando siempre el peor de los casos con un calendarizador adversario, lo cual resulta muy costoso.

El objetivo de la investigación es demostrar que el algoritmo de Xu y Srimani (Xu y Srimani, 2005) para elegir un líder requiere en promedio mucho menos pasos que $O(n^4)$.

Para lograr esto se propone el calendarizador justo. Se analiza el desempeño del algoritmo en diferentes árboles con topologías muy estudiadas en la literatura utilizando el calendarizador propuesto y se comprueba que en promedio se requieren $O(n \log^2 n)$ pasos para estabilizar dichos árboles.

Adicionalmente, se proponen cuatro métodos diferentes para generar árboles arbitrarios donde se utiliza el calendarizador justo y así estudiar el desempeño del algoritmo en estos árboles. Aqui también se demuestra que en promedio se requieren $O(n \log^2 n)$ pasos para estabilizar estos árboles, en la mayoría de los casos.

También se propone una modificación del algoritmo para permitir que los nodos ejecuten sus operaciones en paralelo, de esta manera el algoritmo se acelera considerablemente y el calendarizador se elimina por completo. Se demuestra que para este caso se requiren en promedio $O(k \log n)$ pasos para estabilizar un árbol, donde k es el diámetro del árbol.

El calendarizador justo funciona utilizando procedimientos aleatorios. En este trabajo de tesis se demuestra que los resultados obtenidos con el calendarizador justo se logran con alta probabilidad.

Palabras clave: Auto-Estabilización, Calendarizador, Elección de Líder, Árbol

ABSTRACT of the thesis presented by **Juan Paulo Alvarado Magaña** as a partial requirement to obtain the MASTER OF SCIENCE degree in COMPUTER SCIENCE. Ensenada, Baja California, Mexico. August 2006.

ANALYSIS OF SELF-STABILIZING ALGORITHMS FOR LEADER ELECTION

Abstract approved by:

Dr. José Alberto Fernández Zepeda
Thesis Director

Self-stabilization is a paradigm for fault tolerance in which a distributed system can reach a global valid state from any arbitrary invalid state in a finite number of steps.

The research topic of this thesis is self-stabilizing algorithms for leader election in a tree. Choosing a leader in any graph is to assign unique attributes to one specific node of the graph. A distributed system can loose its leader by suffering a disturbance, it is the obligation of the self-stabilizing algorithm to detect the error and immediately choose a new leader.

Xu and Srimani's self-stabilizing algorithm for leader election (Xu y Srimani, 2005) is the main research topic of this thesis. This algorithm can choose a leader in $O(n^4)$ steps (where n is the total number of nodes in the tree) with an adverse daemon (worst case), this execution time is very costly.

The objective is to demonstrate that Xu and Srimani's algorithm requires on average less then $O(n^4)$ steps.

To reach this objective a fair daemon is proposed. The algorithm's performance on typical tree configurations is analyzed using the fair daemon and it is proven that on average, the algorithm uses $O(n \log^2 n)$ steps to stabilize.

Additionally, four methods to generate arbitrary tree graphs are proposed, where the fair daemon is also used and the performance of the algorithm is analyzed. It is also proven that on average the algorithm uses $O(n \log^2 n)$ steps on many of these cases.

Some modifications to the algorithm are also proposed to allow the nodes to execute their operations in parallel. By doing so, the execution time of the algorithm is greatly improved and the daemon is completely eliminated. It is proven that for these cases, the algorithm uses on average $O(k \log n)$ steps to stabilize a tree of diameter k.

The fair daemon uses a probabilistic method. It is proven in this thesis that the results given by the fair daemon occur with a high probability.

Keywords: Self-Stabilization, Daemon, Leader Election, Trees

Dedicatoria

A mis padres.
A mis maestros.
A Mariamor.

Agradecimientos

A mi asesor, Dr. José Alberto Fernández Zepeda.

A mi comité de tesis, Dr. Carlos Alberto Brizuela Rodríguez y
Dr. Luis Alejandro Márquez Martínez.

Al Centro de Investigación Científica y de Educación Superior de Ensenada.

Al Consejo Nacional de Ciencia y Tecnología.

Contenido

Contenido (continuación)

Capítulo Página

Contenido (continuación)

Capítulo Página

Lista de Figuras

Lista de Tablas

Capítulo I

Introducción

A lo largo de la historia de la computación, muchos investigadores han trabajado para hacer de la computadora una herramienta confiable. Hoy en día la computadora se utiliza para realizar inumerables actividades y conforme avanza la tecnología la computadora se convierte cada vez más en una herramienta indispensable. Gracias a la amplia distribución de las redes de computadoras, muchas de las actividades de cómputo se realizan de forma remota, por esta razón es importante garantizar su funcionamiento correcto.

Como una opción más para aumentar la confiabilidad en los sistemas de cómputo se introduce el concepto de auto-estabilización. Con la auto-estabilización, un sistema, al detectar algún error en su funcionamiento, puede tomar acciones adecuadas para corregirlo en un número finito de pasos, sin la interferencia de algún ente externo. La auto-estabilización es un paradigma de tolerancia a fallas y su estudio es el objetivo del presente trabajo de tesis.

Este capítulo introduce el concepto de algoritmos auto-estabilizantes. En la sección I.1 se presenta una definición de la auto-estabilización. La sección I.2 profundiza en el tema y define a los sistemas y algoritmos auto-estabilizantes; esta sección es la más extensa del capítulo y se divide en subsecciones donde se presentan las características más importantes de este tipo de algoritmos. En la sección I.3 se presenta las aplicaciones que tienen los algoritmos auto-estabilizantes. En la sección I.4 se muestra una breve lista de algunas de las investigaciones previas que se han hecho sobre auto-estabilización. El objetivo de la investigación se define en la sección I.5. La sección I.6 explica la panorámica de la tesis, es decir, las actividades de investigación que se realizaron para elaborar esta tesis. En la sección I.7 se da un resumen de los resultados obtenidos de esta investigación. Finalmente, la sección I.8 muestra la importancia de esta investigación.

I.1 Auto-estabilización

En 1974 Dijkstra (Dijkstra, 1974) introdujo el concepto de Auto-Estabilización como un paradigma para el diseño de algoritmos distribuidos tolerantes a fallas. Dijkstra define a un sistema auto-estabilizante como aquel que, independientemente de su estado inicial, siempre puede llegar a un estado legítimo en un número finito de pasos. La auto-estabilización se diferencia de otros enfoques de sistemas confiables en el siguiente aspecto: En lugar de utilizar métodos específicos para corregir diferentes tipos de fallas con métodos como la replicación, almacenamiento seguro o "backtracking", la auto-estabilización es un mecanismo en donde el sistema está en constante movimiento hacia adelante (el sistema nunca se recupera utilizando información previa respaldada). Dijkstra también observó que la complejidad para lograr esta propiedad en sistemas distribuidos radica en que un nodo solamente puede tomar acciones hacia la estabilización utilizando información local. Por lo tanto, las acciones locales deben lograr, de alguna manera, la estabilización global.

Originalmente el trabajo de Dijkstra tuvo muy pocas aplicaciones, pero rápidamente se convirtió en un enfoque unificado y formal hacia el diseño de sistemas distribuidos tolerantes a fallas.

I.2 Sistemas Auto-Estabilizantes

Los algoritmos auto-estabilizantes han despertado mucho interés, especialmente en los sistemas distribuidos, debido a la preocupación emergente por lograr mayor escalabilidad y distribución aún más extrema en los sistemas de control. En esta sección se definen algunos conceptos básicos y características que tienen los algoritmos auto-estabilizantes.

Un sistema distribuido se puede modelar como un grafo que contenga un número finito de nodos. Si cada nodo almacena un conjunto definido de variables, cada combinación

de valores que estas variables pueden tomar especifica un *estado*. Un *estado global* es una combinación específica de los valores de las variables en todos los nodos del grafo. Un *estado local* es una combinación específica de los valores de las variables en un nodo específico. Un estado es *válido* si los nodos satisfacen un conjunto predefinido de condiciones, en caso contrario se dice que el estado es *inválido*.

Por ejemplo, si existen tres nodos, y cada nodo almacena una variable binaria, entonces existen ocho posibles estados globales, mientras que sólo existen dos estados locales por nodo. Para este ejemplo, si se define como estado válido aquel en el que sólo existe un nodo cuya variable toma el valor de uno, entonces existen tres estados globales válidos y el resto son estados inválidos.

Un sistema distribuido puede estar expuesto a perturbaciones que alteren el estado local de cada nodo, y por consecuencia llevar al sistema a un estado global inválido. El objetivo de un algoritmo auto-estabilizante es el de alcanzar un estado global válido en un número finito de pasos. Un sistema auto-estabilizante corrige errores de transición, o sea, errores que perturben el estado del sistema pero no la funcionalidad de éste. En este trabajo se supone que el estado de un sistema es perturbable, su funcionalidad no.

Schneider (Schneider, 1993) propone la siguiente definición formal para sistemas auto-estabilizantes:

Definición 1. Dado un sistema S donde las condiciones del estado global válido se definen en la regla P, S es auto-estabilizante con respecto a P si satisface las siguientes dos propiedades:

1. Iniciando desde un estado global arbitrario, se garantiza que S llega a un estado global donde P es verdadero, utilizando un número finito de pasos.

2. Una vez que P es verdadero, P no se puede convertir en falso.

En otras palabras, lo que Schneider quiere decir es para que un sistema se considere auto-estabilizante debe haber un estado en el que el sistema esté estable y este sistema debe ser capaz de llegar a dicho estado a partir de cualquier estado inicial. Un sistema auto-estabilizante tiene principalmente dos propiedades que lo hace útil:

1. No se necesita inicializar. Dado que al finalizar la ejecución del algoritmo el estado será válido no es necesario preocuparse por la inicialización correcta del sistema.

2. Puede recuperarse de errores de transición. El sistema después de sufrir una perturbación es capaz de llegar a un estado estable.

I.2.1 Estructura de los algoritmos auto-estabilizantes

Para representar el pseudo-código de un algoritmo auto-estabilizante, los autores tradicionalmente utilizan reglas de la forma *If-Else*. En la parte *If* de estas reglas es donde se comparan las variables locales con respecto a las variables de los vecinos para determinar la validez del estado. Se supone que cada nodo del grafo cuenta con un conjunto finito de variables, además cada nodo es capaz de leer las variables de sus vecinos, de esta forma los nodos determinan si el estado local es válido o no.

Cuando un nodo se da cuenta que está en un estado local inválido se dice que es un nodo candidato, en otras palabras, el nodo encontró un error en su estado local y por lo tanto debe ejecutar acciones para corregirlo.

Por ejemplo, un algoritmo sencillo para colorear los nodos en un grafo puede tener una regla de la siguiente forma: *If (c(u) = c(v)) then c(u) = c(v) + 1*

En esta regla los nodos u y v son vecinos, $c(u)$ es el color del nodo u y $c(v)$ es el color del nodo v. Si el algoritmo encuentra que ambos tienen el mismo color (esto es un

coloreado inválido) el color del nodo u se hace uno más grande que su vecino para cambiar o "corregir" su color.

También se puede apreciar con el ejemplo que ambos nodos, tanto u como v, pueden privilegiarse simultáneamente bajo el esquema de un sistema distribuido, esto podría ocasionar que ambos nodos se recoloreen con el mismo valor, dando como resultado otro coloreado inválido. Este es uno de varios problemas que se deben tener en cuenta al diseñar algoritmos auto-estabilizantes, hay propiedades que estos algoritmos deben tener para garantizar la estabilización y éstas se explican en la siguiente sección.

I.2.2 Propiedades necesarias para obtener auto-estabilización

En esta sección se explica brevemente cuáles son las propiedades que deben encontrarse en un algoritmo auto-estabilizante para que lleve al sistema a un estado válido.

Para que pueda existir la auto-estabilización en un sistema, algunos autores consideran la existencia de un "calendarizador". El *calendarizador* se encarga de coordinar la activación de los nodos dentro del sistema distribuido, su tarea es determinar cuáles de los nodos candidatos pueden modificar sus variables de acuerdo a alguna regla. El calendarizador se necesita para evitar que el proceso de estabilización de un nodo interfiera con los procesos de sus vecinos. La Figura 1 muestra cómo el calendarizador debe escoger entre dos posibles nodos candidatos. Los nodos grises son candidatos para cambiar su color por que detectan que existe un coloreado inválido, el calendarizador escoge uno de ellos y lo vuelve negro. Cuando el calendarizador escoge a un nodo candidato se dice que lo *privilegió*.

Un tipo de calendarizador frecuentemente utilizado es el llamado calendarizador central (Dijkstra, 1974; Xu y Srimani, 2005; Schneider, 1993; Huang, 1993; Antonoiu y Srimani,

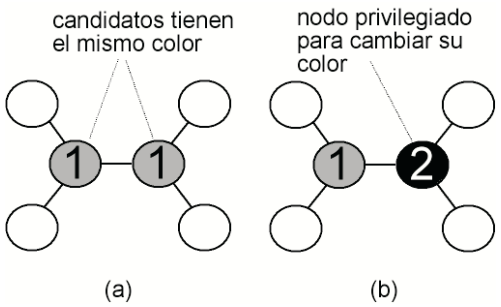

Figura 1: Función del calendarizador. (a) Dos nodos vecinos con el mismo color. (b) El calendarizador escoge el nodo derecho (lo privilegía) y cambia su color

1996; Ghosh y Gupta, 1996; Fich y Johnen, 2001; Burns, 1989). El *calendarizador central* permite a todos los nodos privilegiados ejecutar las tareas de estabilización localmente uno a la vez. Otro calendarizador utilizado en algunos trabajos es el *calendarizador síncrono* (Schneider, 1993; Goddard *et al.*, 2005; Gradinariu y Tixeuil, 2000; Antonoiu y Srimani, 1997), Xu y Srimani (Xu y Srimani, 2005) en su trabajo mencionan el calendarizador síncrono pero no aportan ningún análisis de su algoritmo. El calendarizador síncrono escoge un subconjunto de nodos candidatos para cambiar sus variables sin ocasionar problemas con sus nodos vecinos; por ejemplo, en el caso de un algoritmo auto-estabilizante para el coloreado de grafos, un calendarizador síncrono privilegía a un conjunto de nodos siempre y cuando éstos no sean adyacentes.

La simetría en un sistema es un factor que puede dificultar el diseño de algoritmos auto-estabilizantes. Si existe simetría dentro de un sistema distribuido (esto es, que todos los nodos sean iguales, que todos tengan el mismo número de conexiones, que ninguno de ellos tenga un indentificador único, etc.) es imposible diferenciar un nodo de otro. Un algoritmo auto-estabilizante no puede permitir que los nodos cambien sus variables arbitrariamente ya que podría ocasionar que el sistema no converja por que los nodos siempre estarían cambiando de estado uniformemente manteniendo el estado inválido en cada transición. Para romper esta simetría, se puede utilizar un calendarizador central. Otras soluciones para romper simetría consisten en utilizar asimetría por estado o asimetría por identidad (Schneider, 1993). Un sistema es asimétrico por estados cuando todos los nodos son

idénticos pero tienen diferentes estados locales iniciales. Un sistema es asimétrico por identidad cuando todos sus nodos son idénticos, pero tienen un identificador local único, los sistemas que no utilizan identificadores se les llama anónimos. Otra forma adicional de romper simetría es el uso de algoritmos aleatorios, en donde los nodos deciden si actúan o no dependiendo de una probabilidad dada. Gradinariu y Tixeuil presentan ejemplos sencillos de diferentes formas para romper simetría (Gradinariu y Tixeuil, 2000).

Si un estado global inválido es también un estado final (esto es, un estado del cuál el sistema no puede salir), no habrá estabilidad (Schneider, 1993); a esto se le llama la propiedad de terminación, que es un factor que previene la auto-estabilización. Aunque la propiedad de terminación es muy natural cuando se trata de algoritmos cuyas metas son cuantitativas (por ejemplo, calcular el valor de una función), no es natural cuando se considera dentro del dominio de sistemas distribuidos en donde los procesos no tienen terminación por diseño pero tienen metas como la coordinación y el control. Una forma de terminación en un sistema distribuido ocurre cuando el sistema se bloquea y todos los nodos esperan un evento que nunca llegará, lo cual impide la auto-estabilización.

El aislamiento también es otro factor que previene la auto-estabilización. En los sistemas auto-estabilizantes cada nodo toma acciones con información local para lograr estabilización global, esto hace probable que un nodo llegue a un estado válido local pero aún así el estado global siga siendo no válido. Si un nodo encuentra que su estado local es válido siendo que el sistema en general aún no es estable, este nodo puede aislarse de sus vecinos y no participar en las acciones de estabilización que aún deben tomarse. En este caso el sistema no puede estabilizarse debido a una mala comunicación y coordinación. También las configuraciones similares interfieren con la auto-estabilización en un sistema, es decir, si en el mismo sistema dos estados diferentes son similares de tal forma que no se pueden diferenciar uno del otro y algunos son inválidos, no se puede garantizar la convergencia del sistema.

I.2.3 Comprobación de convergencia

Es importante notar que los algoritmos auto-estabilizantes son sencillos de representar en pseudo-código y el lector los puede interpretar fácilmente. Existen dos factores importantes en el análisis de este tipo de algoritmos.

- Comprobar la convergencia, es decir, demostrar que el algoritmo se estabiliza en un número finito de pasos, tomando en cuenta que se respetaron los puntos mencionados en la sección anterior al momento de diseñar un algoritmo auto-estabilizante.

- Calcular el tiempo de ejecución del algoritmo, como el estado inicial siempre se considera arbitrario, algunas veces resulta complejo predecir con certeza cómo se comporta este tipo de algoritmos. Además, puede darse el caso que el tiempo de ejecución de un algoritmo auto-estabilizante sea muy costoso, especialmente si en lugar de encontrar un tiempo de ejecución exacto, por simplicidad se propone una cota superior para el peor de los casos del algoritmo.

Para comprobar que un algoritmo auto-estabilizante efectivamente converge a un estado estable, muchos autores hacen uso de funciones variantes, por ejemplo (Kessels, 1988; Dijkstra, 1986; Huang, 1993). En otras palabras, definir una función que se evalue en cada transición del sistema (tomando en cuenta variables locales y demás información que el algoritmo utilice). Conforme el algoritmo se ejecuta, la función disminuye su valor en cada paso. Si se comprueba que la función es mínima en un estado válido, entonces se puede comprobar la convergencia del sistema demostrando que la función en efecto disminuye monotónicamente hasta encontrar el valor mínimo. Definir esta función no es trivial, ya que se requiere de un conocimiento profundo del algoritmo para poder modelar su comportamiento en una función discreta.

I.3 Importancia y Aplicaciones

La auto-estabilización es una característica deseada en los sistemas distribuidos tolerantes a fallas. Como éstos no tienen una memoria global compartida, la sincronización global se debe obtener en tiempo finito sin la intervención de agentes externos.

Históricamente, algunos investigadores han intentado resolver el problema de la tolerancia a fallas combatiendo específicamente diferentes fenómenos particulares y encontrar la manera de contrarrestar dichos errores con modelos diseñados precisamente para resolverlos. Por ejemplo, los errores en inicialización, los errores de transmisión, las fallas en procesos, las fallas en memoria, etc. Estos errores se resuelven tradicionalmente uno por uno con métodos específicos y diferentes para cada caso, sin embargo, todos estos errores tienen una solución en común, la de auto-estabilización.

Tradicionalmente las aplicaciones de los algoritmos auto-estabilizantes se enfocan a sistemas distribuidos, y gracias a la facilidad de modelar los sistemas distribuidos utilizando grafos, muchos investigadores trabajan con algoritmos auto-estabilizantes para solucionar problemas en grafos. Entre los problemas de grafos más estudiados existe la elección de líder y el coloreado de grafos, estos dos a su vez tienen diferentes aplicaciones más específicas que se presentan en esta sección.

La elección de líder consiste en un hecho sencillo. Dado un conjunto de nodos candidatos, se debe escoger uno solo para que sea líder, todos los nodos deben estar de acuerdo con la elección y este líder toma características únicas que lo diferencien del resto de los nodos. La Figura 2 muestra un árbol dirigido, todos los nodos apuntan hacia la dirección del líder, el nodo que se eligió como líder es el sombrado y es el único que apunta a sí mismo.

Existen variantes en el problema de elección de líder que dependen de las suposiciones hechas en cada caso. Por ejemplo, se puede suponer que cada nodo cuenta o no con

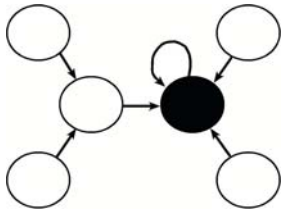

Figura 2: Todos los nodos del grafo saben que el nodo sombreado es el líder

un identificador único. Se puede suponer que se conoce o no cuántos nodos candidatos existen. También se puede suponer que se conoce o no alguna cota superior o inferior para el número de candidatos.

Los algoritmos de elección de líder, en particular, tienen muchas aplicaciones en sistemas distribuidos. Por ejemplo, en los protocolos de comunicación entre grupos, un nuevo coordinador de grupo se debe elegir cuando el coordinador viejo deja de funcionar o si éste deja al sistema. También se utiliza para formar "clusters" dentro de una red de comunicación (Schneider, 1993).

Recientemente ha existido mucho interés en utilizar la elección de líder en ambientes inalámbricos, ya sea para coordinación de rutas, coordinación de sensores y control en general. Los líderes se eligen para controlar el tráfico de mensajes y el ruteo de estos. En los ambientes inalámbricos la elección de líder se convierte en un componente crítico del sistema distribuido, ya que los movimientos frecuentes de usuarios ocasionan que se elijan líderes constantemente (Ghosh, 2001; Schneider, 1993).

La seguridad también es una aplicación de interés para la elección de líder. En arquitecturas de grupos seguros, frecuentemente se escoge a un líder para distribuir las actualizaciones de llaves a todos los miembros del grupo. La elección de líder también es útil en un esquema de replicación y respaldo primario, todas las solicitudes se dirigen hacia el líder donde se procesan y se regresan al origen, el líder actual debe mantener las copias actualizadas en caso de que éste deje de funcionar y un líder nuevo sea electo (Schneider, 1993).

El coloreado de nodos en un grafo es otro problema que se ha estudiado mucho en el área de algoritmos auto-estabilizantes. En este problema, cada nodo tiene una variable, generalmente un número entero positivo, el cual representa un color. Se dice que el coloreado de un grafo es válido cuando cada nodo tiene un color diferente a todos los colores de sus vecinos, es decir, dos vecinos no pueden tener el mismo color. En el grafo que se muestra en la Figura 3, la etiqueta (o color) de cada nodo es diferente a la de todos sus vecinos, por lo tanto es un coloreado válido.

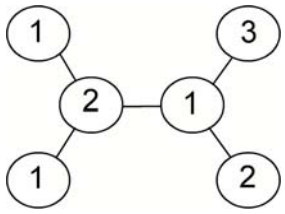

Figura 3: Coloreado válido de un grafo, no hay nodos vecinos con el mismo color

Muchos problemas de calendarización involucran a un número de restricciones por pares para tareas que se pueden ejecutar simultáneamente. Por ejemplo, en la construcción de un horario de clases en una universidad, dos cursos por el mismo maestro no se pueden programar para la misma hora, por otro lado, dos cursos que son requeridos por el mismo grupo de alumnos tampoco pueden programarse para la misma hora. El problema de determinar el número mínimo de intervalos de tiempo necesarios sujetos a estas restricciones es un problema de coloreado de grafos.

Asignar frecuencias a radios móviles también puede modelarse como un problema de coloreado de grafos. En el caso más sencillo, dos usuarios que están a una distancia entre sí relativamente corta no pueden utilizar la misma frecuencia de radio, pero dos usuarios con una distancia suficientemente grande pueden compartir la misma frecuencia sin causar conflictos.

I.4 Trabajo previo

Dijkstra propuso el concepto de auto-estabilización en 1974 en su trabajo titulado "Self-stabilizing systems in spite of distributed control" donde presenta a los sistemas tolerantes a fallas (Dijkstra, 1974).

En 1995, Dolev, Israeli y Moran (Dolev *et al.*, 1995) presentan un algoritmo auto-estabilizante para elección de líder en grafos arbitrarios. Este algoritmo escoge un líder en $O(\Delta D \log n)$ pasos, donde Δ es el grado máximo de un nodo, D es el diámetro de comunicación del grafo y n es el número de nodos en el grafo. El algoritmo funciona de la siguiente manera: primero se eliminan los ciclos del grafo original, esto da como resultado un bosque; el siguiente paso es fusionar aleatoriamente los diferentes árboles en un solo componente, el algoritmo mantiene una relación de padre-hijo entre los nodos que se usa para enraizar este árbol; finalmente la raíz se elige como líder.

En 1996 Antonoiu y Srimani (Antonoiu y Srimani, 1996) trabajaron en algoritmos auto-estabilizantes enfocándose en árboles y en el problema de elección de líder. El problema que ellos enfrentaron en su trabajo inicial fue que no podían escoger como líder a una hoja del árbol de manera anónima (se dice que un sistema es anónimo cuando los nodos no tienen identificadores únicos). Esta situación se ve corregida en sus siguientes investigaciones.

En 1997 Antonoiu y Srimani (Antonoiu y Srimani, 1997) también trabajaron en algoritmos auto-estabilizantes para encontrar el centro de un árbol. El algoritmo usa una regla que dice que para cada nodo u, se debe regresar Verdadero si el nodo u no tiene vecinos con un valor igual a un entero l y tiene cuando mucho un vecino con un valor mayor a l. La forma en la cual este algoritmo encuentra el centro es primero igualar todos los nodos hoja a cero y después para cada nodo u encontrar el mínimo valor del entero l que haga verdadera la regla y tomar éste como su valor. Al final, el nodo con el valor máximo global se considera el centro.

Ghosh y Gupta diseñaron un algoritmo auto-estabilizante en 1996 para elección de líder en anillos bi-direccionales (Ghosh y Gupta, 1996). Este algoritmo converge en tiempo constante. Utilizan identificadores únicos a través del anillo para elegir un líder pero sólo corrige errores si existe exactamente una falla en algún nodo.

En el 2001 Fich y Johnen propusieron un algoritmo auto-estabilizante para elección de líder dentro de un anillo unidireccional (Fich y Johnen, 2001). Este algoritmo converge a un estado global válido en $O(n^2)$ pasos y funciona para anillos cuyo tamaño es un número primo. El algoritmo es similar a "token passing", donde existe un token que viaja a través del anillo, al igual que en la elección de líder sólo puede existir un token; si existen más de un token, se deben destruir los tokens adicionales; similarmente si existe más de un líder, se deben eliminar los líderes que sobren. La manera en que eligen cuál token eliminar es sencilla, primero cada nodo tiene una variable donde se guarda la distancia desde este nodo (en caso que no tenga el token) hasta el nodo que tiene el token (las distancias se calculan en una sola dirección). Si existe un nodo con un token este compara su variable con la del vecino de la izquierda y si encuentra que ésta es más grande, entonces este nodo elimina su token y por lo tanto ya no se puede elegir como líder. Mediante eliminaciones de este tipo se llega a un estado global válido.

En el 2005, Xu y Srimani proponen un algoritmo auto-estabilizante para elección de líder (Xu y Srimani, 2005). Este algoritmo se puede aplicar en árboles arbitrarios, es completamente anónimo (no se utilizan etiquetas para identificar a cada nodo), además, este algoritmo puede elegir un nodo hoja como líder (al contrario del algoritmo de (Antonoiu y Srimani, 1996)). Este es el algoritmo en el que se basa el presente trabajo de tesis, por tal razón se da una explicación más profunda sobre el algoritmo en los capítulos siguientes.

I.5 Objetivos

El objetivo de este trabajo de investigación es proponer un método de análisis que ayude a calcular el tiempo de ejecución promedio del algoritmo de Xu y Srimani (Xu y Srimani, 2005) y comprobar que éste es menor a $O(n^4)$.

I.6 Panorámica de la tesis

Las actividades que se hicieron durante el trabajo de investigación de esta tesis son las siguientes.

1. Se propone el uso de un calendarizador justo para la ejecución del algoritmo de Xu y Srimani (Xu y Srimani, 2005).
2. Se calcula el tiempo promedio de ejecución del algoritmo de Xu y Srimani (Xu y Srimani, 2005) en diferentes tipos de árboles.
3. Se hace un análisis de alta probabilidad para la ejecución del algoritmo de Xu y Srimani.
4. Se proponen cuatro diferentes métodos para generar árboles aleatorios.
5. Se calcula el tiempo de ejecución del algoritmo en cada uno de los tipos de árboles aleatorios que se definieron.
6. Se propone una modificación al algoritmo de Xu y Srimani para permitir que los nodos ejecuten sus operaciones en paralelo.
7. Se calcula el tiempo de ejecución del algoritmo considerando que los nodos hacen sus operaciones en paralelo.

I.7 Resultados obtenidos

En las tablas que se muestran en esta sección se puede apreciar los resultados que se obtivieron durante el trabajo de investigación de esta tesis.

La Tabla I muestra el tiempo promedio de ejecución del algoritmo de Xu y Srimani (Xu y Srimani, 2005) en diferentes árboles de n nodos. Como referencia para el lector, el peor tiempo de ejecución de este algoritmo en general es $O(n^4)$ pasos.

Tabla I: Tiempo promedio de ejecución

Árbol	Diámetro	$T(n)$
Binario Balanceado	$\log n$	$O(n \log^2 n)$
Profundidad Logarítmica Doble	$\log \log n$	$O(n \log^2 n)$
Arbitrario de diámetro $O(\log n)$	$c \log n$	$O(n \log^2 n)$

La Tabla II muestra el tiempo de promedio ejecución del algoritmo de Xu y Srimani (Xu y Srimani, 2005) en cada tipo de árbol generado aleatoriamente por cada uno de los métodos definidos.

Tabla II: Tiempo promedio de ejecución en árboles arbitrarios

Árbol	Diámetro	$T(n)$
Tipo 1	$O(\log n)$	$O(n \log^2 n)$
Tipo 2	$O(n^{\frac{1}{2}})$	$O(n^{\frac{3}{2}} \log n)$
Tipo 3	$O(\log \log n)$	$O(n \log^2 n)$
Tipo 4	$O(\log n)$	$O(n \log^2 n)$

La Tabla III muestra el tiempo de promedio ejecución del algoritmo de Xu y Srimani (Xu y Srimani, 2005) considerando que los nodos ejecutan sus operaciones de estabilización en paralelo.

Tabla III: Tiempo promedio de ejecución con operaciones en paralelo

Árbol	Diámetro	$T(n)$
Binario Balanceado	$\log n$	$O(\log^2 n)$
Profundidad Logarítmica Doble	$\log \log n$	$O(\log n \log \log n)$
Arbitrario de diámetro k	$O(k)$	$O(k \log n)$

I.8 Contribuciones de la tesis

El algoritmo de Xu y Srimani (Xu y Srimani, 2005) requiere de $O(n^4)$ pasos para elegir un líder, lo cual es un tiempo de ejecución muy costoso, esto es por que se calculó tomando en cuenta el peor caso posible.

Se comprueba que utiliza el calendarizador justo que se propone el algoritmo en promedio es mucho más rápido.

Otra aportación principal es el método de análisis que se utilizó. Este método tiene la ventaja de que se puede aplicar a cualquier tipo de árbol, como se explica en los siguientes capítulos. Además, el tiempo de ejecución que se obtiene con este método es el promedio y no el peor de los casos.

Los métodos que se definen para generar árboles aleatorios son importantes porque a través de estos se puede predecir fácilmente la topología de un árbol, y por lo tanto, poder así calcular el tiempo de ejecución del algoritmo.

Adicionalmente se demuestra cómo es posible modificar el algoritmo para permitir ejecuciones en paralelo y eliminar por completo el calendarizador, para así acelerar el tiempo de ejecución aún más.

I.9 Organización de la tesis

La tesis se ha organizado de la siguiente manera. El Capítulo II se dedica al algoritmo auto-estabilizante de Xu y Srimani (Xu y Srimani, 2005). Aquí se explica su funcionamiento y el análisis que hicieron sus autores para calcular su tiempo de ejecución. El Capítulo III trata sobre el calendarizador justo que se propone y cómo éste influye en el desempeño del algoritmo. En este mismo capítulo también se analizan diferentes tipos de árboles y se calcula el tiempo de ejecución en cada uno de ellos utilizando el calendarizador justo. En el Capítulo IV se proponen diferentes métodos para generar árboles aleatorios y se calcula el tiempo de ejecución esperado para cada método utilizando también el calendarizador justo. En el Capítulo V se propone una modificación al algoritmo de Xu y Srimani (Xu y Srimani, 2005) para permitir que se ejecute en forma paralela y se calcula el tiempo de ejecución promedio considerando para este caso. El Capítulo VI presenta las concluciones a las que se llegó al finalizar este trabajo de investigación. Finalmente, el Capítulo VII propone algunas líneas de investigación relacionadas con este trabajo.

Capítulo II

Algoritmo Auto-Estabilizante para elección de líder de Xu y Srimani

Este capítulo explica cómo funciona el algoritmo auto-estabilizante de Xu y Srimani (2005) para elección de líder. Se explican las tres reglas que usa el algoritmo y el análisis que hicieron sus autores para calcular su tiempo de ejecución.

II.1 El algoritmo

En la elección de líder, todos los nodos en un grafo escogen un nodo líder, el cual toma propiedades específicas que lo diferencian de los demás y además todos los nodos del grafo están de acuerdo en quien es el líder. El algoritmo de Xu y Srimani (2005) resuelve este problema para un árbol arbitrario.

Cada nodo i del árbol, donde $1 \leq i \leq n$ cuenta con dos variables. La primera variable x_i toma un valor entero positivo; la segunda variable es el apuntador P_i que se dirige o apunta hacia un vecino del nodo i.

El algoritmo encuentra al líder a través de la ejecución sucesiva de tres reglas. Cada nodo del árbol utiliza estas tres reglas para determinar las operaciones a ejecutar. Las reglas son las siguientes:

1. R_1: si el nodo i tiene dos o más vecinos j con un valor x_j mayor a x_i, entonces el nodo i puede ejecutar las operaciones de estabilización de la regla R_1. La operación de estabilización es la siguiente. La variable x_i se convierte en un máximo local, o

sea, se hace estrictamente uno más grande que el máximo de los valores x_j de sus vecinos. Además, el apuntador P_i se iguala a i, o sea, el nodo i apunta a sí mismo.

Nota: en el trabajo original los autores no aclaran explícitamente lo que sucede cuando $x_i = x_j$, pero esto se aclara en el Capítulo III.

Ejemplo 1: en la Figura 4a se ve cómo el nodo sombreado puede ejecutar las operaciones de la regla R_1 ya que tiene dos vecinos más grandes. La Figura 4b muestra cómo el nodo sombreado se convierte en el máximo local y además su apuntador (representado por un arco) apunta a sí mismo.

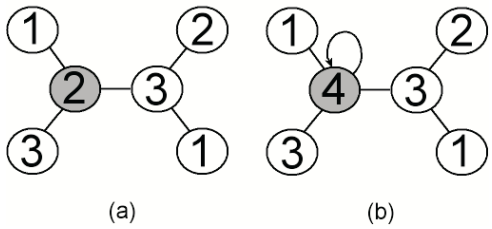

Figura 4: Las operaciones de estabilización de R_1 convierten al nodo sombreado en máximo local

2. R_2: si el nodo i tiene solamente un vecino j con $x_j > x_i$, entonces el nodo i puede ejecutar las operaciones de estabilización de la regla R_2. La operación de estabilización es la siguiente. El valor de la variable x_i se convierte en exactamente uno menos que el valor x_j y el apuntador P_i apunta a j.

   Nota: en el trabajo original los autores no aclaran explícitamente lo que sucede cuando $x_i = x_j$, pero esto se aclara en el Capítulo III.

   Ejemplo 2: en la Figura 5a los nodos sombreados pueden ejecutar las operaciones de la regla R_2 por que tienen exactamente un vecino mayor. La Figura 5b muestra cómo los nodos sombreados ajustan su valor al máximo local.

3. R_3: si el nodo i no tiene vecinos j con un valor $x_j > x_i$ y si el nodo i no está apuntado hacia sí mismo, entonces el nodo i puede ejecutar las operaciones de estabilización

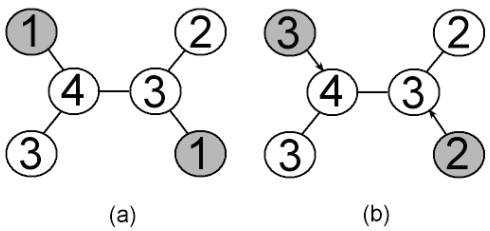

Figura 5: Las operaciones de estabilización de R_2 ajustan los nodos sombreados al máximo local

de la regla R_3. La operación de estabilización es la siguiente. El apuntador P_i se iguala a i, o sea, el nodo i apunta a sí mismo.

Ejemplo 3: en la Figura 6a el nodo sombreado puede ejecutar las operaciones de la regla R_3 por que no tiene vecinos mayores y no apunta a si mismo. La Figura 6b muestra cómo el nodo sombreado apunta a sí mismo después de ejecutar las operaciones de la regla R_3.

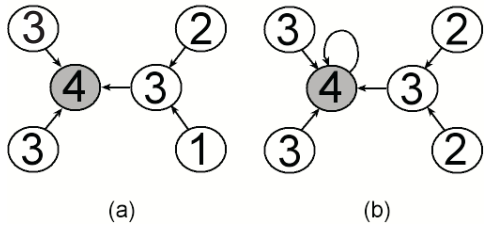

Figura 6: El nodo sombreado ejecuta las operaciones de R_3 para apuntar a sí mismo

Cuando ningún nodo cumpla con alguna de las condiciones de las reglas, se dice que el árbol se ha estabilizado. El líder electo es el nodo que tiene el valor x más grande (máximo global), mientras que el resto de los nodos apuntan hacia la dirección del líder. Si i es el nodo líder, la diferencia entre x_i y x_j, para toda $j \neq i$ es igual a la distancia que existe entre el nodo i y el nodo j.

II.2 Análisis de complejidad

El análisis de este algoritmo supone la existencia de un calendarizador adversario. El calendarizador tiene la tarea de decidir cuál de todos los nodos privilegiados puede hacer un movimiento. Un calendarizador adversario siempre escoge el movimiento que resulte más costoso para el algoritmo. Para calcular el tiempo de ejecución del algoritmo, los autores toman en cuenta el peor caso posible. Ellos suponen que todos los nodos del árbol cumplen con las condiciones de la regla R_1 y calculan cuántas veces se podrían ejecutar las operaciones de esta regla sin tomar en cuenta las otras dos.

En el algoritmo de Xu y Srimani (2005) definen una partición del conjunto de nodos denotada como $\{S_0, S_1, ..., S_m\}$ donde S_0 es el conjunto de nodos hoja del árbol original y para cada S_i donde $i > 0$, los nodos que pertenecen a S_i son los nodos hoja del subárbol que resulta cuando se eliminan todos los nodos que se encuentran en los conjuntos $S_{i-1}, S_{1-2}, ..., S_0$.

Parte del procedimiento del análisis de complejidad que se hace en su artículo consiste en definir cuántas veces se ejecutan las operaciones de la regla R_1.

Las operaciones de la regla R_1 convierten el valor de x_u en el máximo local, y ésta es la única regla que incrementa el valor del máximo local. Para que un mismo nodo u ejecute las operaciones de la regla R_1 dos veces consecutivamente al menos dos de sus vecinos deben ejecutar las operaciones de la regla R_1 también.

Todos los nodos que se encuentran en el conjunto S_{m-1} tienen vecinos en el conjunto S_m. Un nodo hoja del árbol original (esto es, un nodo del conjunto S_0) nunca es candidato de R_1 por que sólo tiene un vecino.

Dado un nodo i, sea $N(i)$ el conjunto de vecinos de i, definen la variable a_i como las

veces que el nodo i ejecuta las operaciones de la regla R_1, donde

$$a_i \leq 1 + \sum_{j=1}^{N(i)} a_j \tag{1}$$

esto es, las veces que el nodo i ejecuta las operaciones de la regla R_1 es cuando mucho uno más las veces que todos sus vecinos ejecutan las operaciones de la regla R_1. También definen A_m como el número total de veces que se ejecutan las operaciones de la regla R_1 por los nodos de S_m.

$$A_m = \sum_{i \in m} a_i \tag{2}$$

El número total de veces que los nodos de S_m ejecutan las operaciones de la regla R_1 está acotado por el número total de veces que los nodos de S_{m-1} ejecutan las operaciones de la regla R_1 más el tamaño de S_m, de tal manera que $A_m \leq A_{m-1} + |S_m|$.

El número total de veces que los nodos de S_m ejecutan las operaciones de la regla R_1, para $m > 0$, es cuando mucho n, $A_m \leq \sum_{k=1}^{m} S_k \leq n$. Así ellos concluyen que el número de veces que se ejecutan las operaciones de la regla R_1 es $O(n^2)$ dado que $\sum_{m=0}^{n} A_m \leq n + \frac{n^2}{2}$.

El siguiente paso del análisis es calcular las veces que se ejecutan las operaciones de la regla R_2. En esta parte del análisis se supone que todos los nodos cumplen con las condiciones de la regla R_2 y que ningun candidato de otra regla se privilegía. Se sabe que la regla R_2 no modifica el valor del máximo local. Xu y Srimani (Xu y Srimani, 2005) dicen que un nodo a distancia 1 del máximo global ejecuta las operaciones de la regla R_2 una vez cuando mucho. Un nodo a distancia k del máximo global ejecuta las operaciones de la regla R_2 k veces. Por lo tanto, si ningún nodo es candidato de R_1 o R_3, las operaciones de la regla R_2 se ejecutan cuando mucho $\frac{n(n+1)}{2} = O(n^2)$ veces. El número de veces que se ejecutan las operaciones de la regla R_2 también es $O(n^2)$, por que cada vez que R_1 modifica el máximo local R_2 debe ajustar el valor de x de los demás vecinos para que la diferencia sea uno.

Si se combinan la veces que se ejecutan las operaciones de la regla R_1 y R_2, se necesitan $O(n^4)$ pasos para elegir un líder. El algoritmo de Xu y Srimani (Xu y Srimani, 2005) se ejecuta en $O(n^4)$ pasos, en el peor de los casos, debido a la interacción que existe entre las reglas R_1 y R_2, las operaciones de la regla R_3 se ejecutan $O(n)$ veces ya que sólo la ejecutan los máximos locales.

Adicionalmente, la prueba de corrección del algoritmo se demuestra en (Xu y Srimani, 2005).

Capítulo III

Análisis del algoritmo utilizando un calendarizador justo en árboles específicos

En este capítulo, el autor de este trabajo de tesis propone un calendarizador justo y analiza cómo éste afecta al desempeño del algoritmo de Xu y Srimani (2005). También se calcula el tiempo de ejecución del algoritmo cuando se aplica en árboles de topologías específicas. Las topologías que se estudian son el árbol binario balanceado, el árbol de profundidad logarítmica doble y un árbol arbitrario de diámetro $O(\log n)$.

III.1 Definiciones

Sea $T = (V, E)$ un árbol no dirigido con $|V| = n$ nodos. Un *nodo candidato* es aquel que cumple con las condiciones de alguna de las reglas de auto-estabilización y está listo para ejecutar las operaciones de estabilización. Un *nodo privilegiado* es aquel nodo candidato que el calendarizador escoge para ejecutar sus operaciones de estabilización.

El *externivel* y el *internivel* de un nodo son valores enteros no negativos que se asignan a él de acuerdo a su posición en el árbol. El externivel de algún nodo hoja es 0, los nodos con externivel 1 son los nodos hoja que resultan cuando se eliminan del árbol todos los nodos con externivel 0, los nodos con externivel 2 son los nodos hoja que resultan cuando se eliminan del árbol todos los nodos con externivel 1 y 0. De tal forma que los nodos con externivel i son los nodos hoja que resultan cuando se eliminan todos los nodos con externivel desde 0 hasta $i - 1$. El nodo con externivel más grande es el centro del árbol. Similarmente, el internivel del nodo i indica la distancia del nodo i a un nodo específico j.

Nota: el concepto de externivel es equivalente a la definición de los conjuntos S_i

en el artículo de Xu y Srimani (Xu y Srimani, 2005). Todos los nodos del conjunto S_i, $0 \leq i \leq m$, tienen externivel i.

La Figura 7 muestra las etiquetas externivel del árbol, mientras que la Figura 8 muestra las etiquetas internivel del árbol cuando se toma como referencia al nodo sombreado.

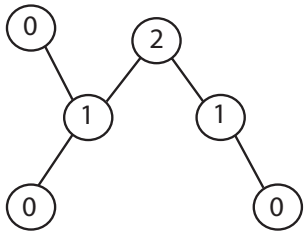

Figura 7: Grafo con etiquetas externivel

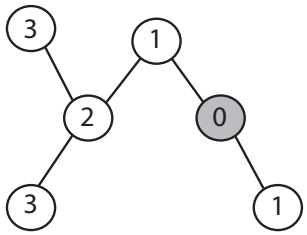

Figura 8: Grafo con etiquetas internivel. El nodo sombreado es el nodo referencia

Ahora se define al *calendarizador*. Desde el punto de vista de sistemas distribuidos y algoritmos para grafos, el *calendarizador* es un ente que se encarga de coordinar la activación de los nodos dentro del sistema distribuido. Entiéndase como activación, al hecho de permitir a un nodo ejecutar cierto tipo de operaciones. Desde el punto de vista de algoritmos auto-estabilizantes, la tarea del calendarizador es determinar cuál (o cuáles) de los nodos candidatos puede (o pueden) privilegiarse. Se utiliza al calendarizador para romper la simetría del sistema.

Un calendarizador usado frecuentemente es el llamado calendarizador central (Dijkstra, 1974; Xu y Srimani, 2005; Schneider, 1993; Huang, 1993; Antonoiu y Srimani, 1996; Ghosh y Gupta, 1996; Fich y Johnen, 2001; Burns, 1989); este calendarizador privilegía

a los nodos candidatos uno a la vez para ejecutar las operaciones de estabilización. Otro calendarizador es el síncrono (Schneider, 1993; Goddard *et al.*, 2005; Gradinariu y Tixeuil, 2000; Antonoiu y Srimani, 1997; Xu y Srimani, 2005). Éste activa un subconjunto de nodos privilegiados para ejecutar sus tareas y supone que ellos pueden ejecutar operaciones de estabilización sin ocasionar interferencias o conflictos con otros nodos.

El calendarizador que Xu y Srimani (Xu y Srimani, 2005) utilizan en su investigación es el llamado calendarizador adversario. Funciona de la misma manera que un calendarizador central, sin embaro tiene la particularidad de que el calendarizador escoge siempre el nodo cuyo movimiento sea el más costoso para el algoritmo. De su análisis, ellos concluyen que su algoritmo encuentra un líder en $O(n^4)$ pasos, para el peor de los casos. La ventaja de suponer un calendarizador adversario es que con él se calcula el peor tiempo de ejecución del algoritmo.

El calendarizador que se propone en el presente trabajo de tesis lo nombramos calendarizador justo. Este calendarizador es una variación del calendarizador central, sin embargo funciona utilizando un proceso probabilístico, como se explica a continuación. El calendarizador asigna a cada nodo del conjunto de candidatos la misma probabilidad para privilegiarse; de esta manera todos los nodos candidatos tienen la misma oportunidad de ejecutar alguna operación de estabilización.

Nuestro modelo de calendarizador ofrece varias ventajas. Por un lado, es un calendarizador cuya implementación es más factible ya que sólo privilegía un nodo al azar (no es realista suponer que el calendarizador sabe el costo de los movimientos). Por otro lado, utilizando este calendarizador, se demuestra que el tiempo promedio de ejecución del algoritmo (sin que éste sufra modificaciones) es mucho menor que $O(n^4)$. El autor supone que el caso promedio es un comportamiento que se apega más a la realidad y da una idea más cercana a cómo se comportaría este algoritmo en la práctica. En este trabajo de tesis el término $\log x$ debe entenderse como el logaritmo de base dos de x y $\ln x$ como

el logarítmo natural de x. Adicionalmente, si la probabilidad de que algún evento ocurra es $1 - \frac{1}{n^\alpha}$ se dice que es con alta probabilidad.

III.2 Estabilización de acuerdo a R_1

La regla R_1 dice que si el nodo i tiene dos o más vecinos j con un valor $x_j > x_i$ entonces $x_i = max\{x_j\} + 1$, esto es, x_i toma el valor máximo de sus vecinos más uno (de esta forma se convierte en un máximo local).

Nota: en el trabajo original (Xu y Srimani, 2005) los autores no aclaran explícitamente que sucede cuando $x_i = x_j$, pero a través del estudio del algoritmo se concluye que funciona adecuadamente si la condición es $x_j \geq x_i$.

Se dice que un árbol es estable con respecto a R_1 si:

1. Ningún nodo es candidato de R_1.
2. Existe un máximo global único, esto es, $x_i > x_j$ para cualquier $j \neq i$.

Como consecuencia de las dos condiciones anteriores, si un árbol T es estable con respecto a R_1 y la variable x_i del nodo i es el máximo global, entonces cualquier nodo j, vecino de i, tiene una variable $x_j < x_i$. En general, cualquier nodo k que se encuentre a una distancia t del nodo i, tiene una variable x_k que es menor a las variables correspondientes de aquellos nodos que se encuentran en la trayectoria que va de k al nodo i. La Figura 9 muestra un árbol estable con respecto a R_1. Note que el nodo con $x_3 = 10$ es el máximo global y además los valores de las variables de los nodos disminuyen conforme estén más retirados del máximo global.

De la misma manera, se dice que un subárbol T_k es estable con respecto a R_1 si se cumple lo siguiente:

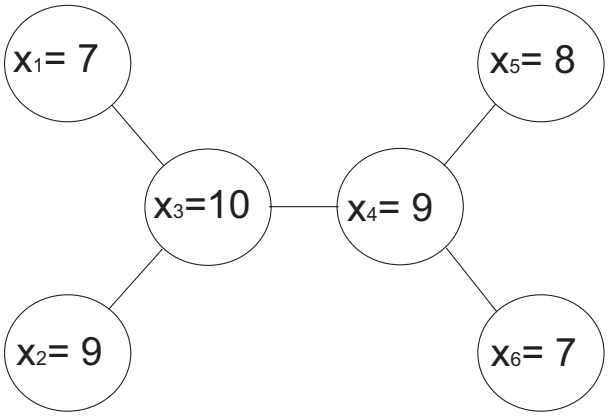

Figura 9: Árbol estable con respecto a R_1

1. T_k es el subárbol que se obtiene al desconectar el nodo k de la arista que lo conecta a su vecino máximo.

2. Cumple con las condiciones de estabilización de acuerdo a R_1 siendo k el máximo global del nuevo subárbol.

En la Figura 10 se muestra cómo el subárbol formado por los nodos sombreados es estable con respecto a R_1. Para este ejemplo el nodo con valor $x_4 = 9$ se desconecta del árbol al remover su arista que lo conecta a su vecino máximo, entonces este nodo es ahora el máximo global de su subárbol.

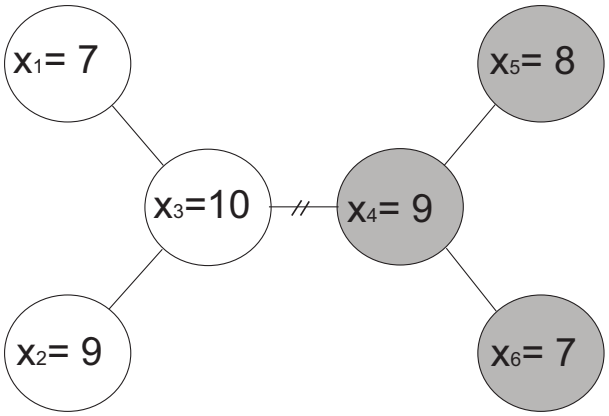

Figura 10: El subárbol sombreado es estable con respecto a R_1

Si se permite al algoritmo privilegiar únicamente a candidatos de R_1, las operaciones se

pueden ejecutar en diversos nodos del árbol, sin tener en general un patrón definido. Sin embargo, después de varias iteraciones, se puede observar que la regla R_1, gradualmente se va ejecutando alrededor de una región muy específica, que eventualmente es el máximo global.

Esta tendencia se puede ver como que el proceso de estabilización empieza en los extremos y se propaga hacia el nodo que eventualmente será máximo global. La posición del máximo global puede ser cualquiera al finalizar la ejecución del algoritmo, ella depende de los valores iniciales de las variables asociadas a los nodos y de la elección de los nodos privilegiados. En la Figura 11 se ilustra cómo es el proceso de estabilización de un árbol con respecto a R_1. La Figura 11a muestra un árbol inicial inestable, los nodos sombreados pueden ejecutar las operaciones de la regla R_1 y el nodo obscuro eventualmente será el máximo global. En la Figura 11b los nodos hoja son los primeros en obtener la estabilización. Las Figuras 11c y 11d muestran cómo eventualmente el proceso de estabilización termina en el máximo global.

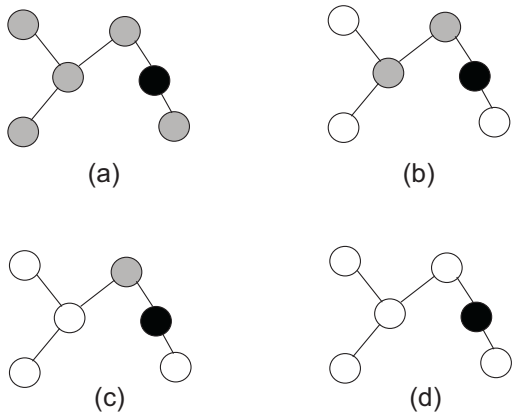

Figura 11: En general, los nodos más lejanos al máximo global son los primeros obtener la estabilización. El proceso termina en el máximo global

Los nodos con externivel 0 están estables con respecto a R_1 ya que tienen únicamente un solo vecino y por lo tanto no pueden ser candidatos de R_1. Cualquier nodo con externivel 1 se estabiliza con respecto a R_1 después de que se ha privilegiado y ha ejecutado la

operación de estabilización de la regla R_1, esto sucede por que una vez que un nodo de este externivel se establece como máximo local ya es estrictamente mayor que sus hijos (hojas) y por más grande que crezca su padre, nunca más será candidato de R_1. De la misma manera, los nodos con externivel i sólo se pueden estabilizar una vez que todos los nodos con externivel menor que i sean estables. Note que para cualquier árbol, el número de nodos en el externivel i es menor o igual al número de nodos en el externivel $i-1$.

Sea y_i la cantidad de candidatos en el externivel i y sea $z_i = \sum_{j=i}^{k} y_j$ el número de candidatos en todos los externiveles desde i hasta k. Para calcular el promedio de pasos necesarios para estabilizar un árbol con respecto a R_1, primero se deben calcular los pasos necesarios para estabilizar un externivel arbitrario. Como se menciona en el párrafo anterior, para estabilizar un externivel i se necesitan estabilizar todos los externiveles $j < i$, por eso se supone que los externiveles $j < i$ ya son estables.

En el árbol existen candidatos de R_1 en el externivel i y en cada externivel $j > i$. Considerando el calendarizador justo, éste puede privilegiar un nodo de i o un nodo de j con la misma probabilidad. La probabilidad para privilegiar algún nodo de i es por lo tanto $\frac{y_i}{z_i}$. Sea k_i una variable aleatoria que representa el número de pasos que se requieren para que un nodo candidato del externivel i se privilegíe. El valor esperado de k_i es $\frac{z_i}{y_i}$ (por tener k_i una distribución geométrica). Una vez que se privilegía algún nodo de i, el número de pasos esperados para privilegiar otro candidato en el externivel i es $\frac{z_i-1}{y_i-1}$. El número de pasos esperados para privilegiar el j-ésimo candidato en el externivel i es $\frac{z_i-j}{y_i-j}$. En total, el número esperado de pasos T_i que se necesitan para privilegiar todos los candidatos del externivel i es:

$$T_i = \sum_{j=0}^{j=y_i-1} \frac{z_i - j}{y_i - j}. \tag{3}$$

$$T_i = \sum_{j=0}^{j=y_i-1} \frac{z_i}{y_i - j} - \sum_{j=0}^{j=y_i-1} \frac{j}{y_i - j} \tag{4}$$

$$T_i = z_i \sum_{j=1}^{j=y_i} \frac{1}{j} - \sum_{j=1}^{j=y_i} \frac{y_i - j}{j} \tag{5}$$

$$T_i = (z_i - y_i) \sum_{j=1}^{j=y_i} \frac{1}{j} + \sum_{j=1}^{j=y_i} \frac{j}{j} \tag{6}$$

La sumatoria $H_n = \sum_{j=1}^{n} \frac{1}{j}$ se conoce como el n-ésimo número armónico y su valor es $\ln n + O(1)$. Por lo tanto.

$$T_i = (z_i - y_i)(\ln y_i + O(1)) + y_i \tag{7}$$

Para una n suficientemente grande, $\ln y_i = o(1)$, entonces la Ecuación 7 se puede simplificar a (8).

$$T_i = (z_i - y_i) \ln y_i + y_i \tag{8}$$

El número esperado de pasos $T(n)$ necesarios para estabilizar con respecto a R_1 a cualquier árbol de n nodos y externivel máximo k está dado por:

$$T(n) = \sum_{i=1}^{k} T_i = \sum_{i=1}^{k} (z_i - y_i) \ln y_i + y_i \tag{9}$$

La Ecuación 9 se puede representar como logaritmo de base dos. Si $\log y_i = \frac{\ln y_i}{\ln 2}$ entonces $\ln y_i = \ln 2 \log y_i$, por lo tanto $T(n)$ es igual a

$$T(n) = \ln 2 \sum_{i=1}^{k} (z_i - y_i) \log y_i + y_i \tag{10}$$

Note que los valores de z_i y y_i dependen de la topología del árbol. A continuación se emplea la Ecuación 10 para calcular el número de pasos promedio para estabilizar diversas topologías.

III.3 Estabilización con respecto a R_1 en un árbol binario balanceado

Suponga que se tiene un árbol binario balanceado de n hojas. Suponga que no se privilegían candidatos de la regla R_2. Sea C_1 el conjunto de nodos candidatos de la regla R_1. En un árbol binario balanceado, todos los nodos internos tienen tres vecinos (excepto la raíz, que tiene dos). Suponga que cada nodo i del árbol, donde $1 \leq i \leq n$, se le asigna a su variable x_i un valor aleatorio (con el fin de simular un sistema inestable). Suponga que el nodo i tiene 3 vecinos, denotados por a_1, a_2 y a_3. Sea X_{a_j}, para $j = \{1, 2, 3\}$, una variable aleatoria indicadora que toma el valor de uno si $x_{a_j} \geq x_i$ y cero de otra forma y sea $X = \sum_{j=1}^{3} X_{a_j}$; la variable aleatoria X representa el número de vecinos de i que tienen variables x_j mayores o iguales a x_i. Entonces para un nodo interno existen cuatro formas, de las ocho posibles, de que al menos dos de sus vecinos sean mayores o iguales (ver renglones 4, 6, 7, y 8 de la Tabla IV).

Por lo tanto la probabilidad de que un nodo interno sea elemento de C_1 es $\frac{1}{2}$. Para el caso de las hojas, la probabilidad de que un nodo hoja sea elemento de C_1 es cero, ya que sólo tienen un vecino y por lo tanto nunca cumple con la condición de la regla R_1. En el caso de la raíz, que sólo tiene dos vecinos, existe una forma, de cuatro posibles, de que sus dos vecinos sean mayores; por lo tanto la probabilidad de que la raíz sea elemento de C_1

Tabla IV: Comparación de la variable x_i del nodo i con respecto a las variables x_j de sus vecinos

X_{a_1}	X_{a_2}	X_{a_3}	$X = \sum_{j=1}^{3} X_{a_j}$
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	2
1	0	0	1
1	0	1	2
1	1	0	2
1	1	1	3

es $\frac{1}{4}$. La probabilidad de que cualquier nodo del árbol binario balanceado se privilegíe se puede acotar por arriba con la probabilidad de que los nodos internos se privilegíen, por lo tanto la probabilidad de que un nodo i sea elemento de C_1 es $Pr\{i \in C_1\} \leq \frac{1}{2}$. La Figura 12 muestra un árbol binario balanceado de n hojas, los nodos hoja tienen exactamante un vecino mientras que la raíz tiene dos, cualquier otro nodo interno tiene tres vecinos. Esta figura también muestra el externivel de cada nodo.

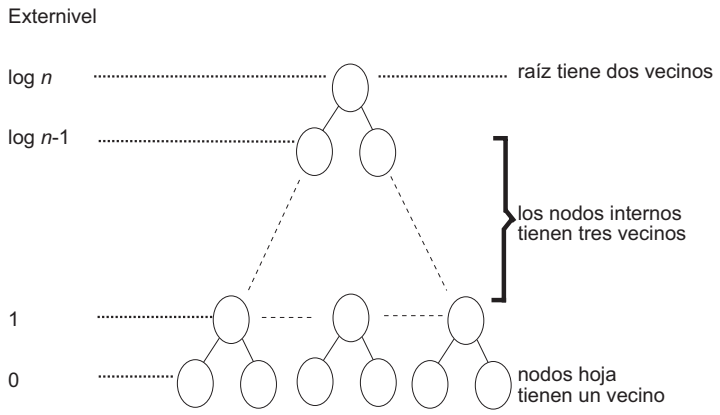

Figura 12: Árbol Binario Balanceado de n hojas

Con esta información se definen los valores de y_i y z_i. El número esperado de candidatos y_i en el externivel i, para $1 \leq i \leq \log n$, es:

$$E[y_i] = \frac{2^{\log n - i}}{2} \tag{11}$$

El número esperado de candidatos z_i en todo externivel $j \geq i$, para $1 \leq i \leq \log n$ es.

$$E[z_i] = \frac{2^{\log n - i + 1} - 1}{2} \tag{12}$$

Usando la Ecuación 10 se puede calcular el número de pasos promedio $T(n)$ necesarios para estabilizar el árbol binario balanceado de n hojas con respecto a R_1.

$$T(n) = \ln 2 \sum_{i=1}^{\lceil \log n \rceil} \left[\left(\frac{2^{\log n - i + 1} - 1}{2} - \frac{2^{\log n - i}}{2} \right) \log \frac{2^{\log n - i}}{2} 2 + \frac{2^{\log n - i}}{2} \right] \tag{13}$$

La sumatoria

$$T(n) = \ln 2 \sum_{i=1}^{\lceil \log n \rceil} 2^{\log n - i} \tag{14}$$

es igual a

$$T(n) = \ln 2 \sum_{i=1}^{\lceil \log n \rceil} 2^{i} \tag{15}$$

Por lo tanto, la Ecuación 13 se simplifica a

$$T(n) = \ln 2 \sum_{i=1}^{\lceil \log n \rceil} \left[\left(\frac{2^{i+1} - 1}{2} - \frac{2^i}{2} \right) \log \frac{2^i}{2} + \frac{2^i}{2} \right] \tag{16}$$

$$T(n) = \frac{1}{2} \ln 2 \sum_{i=1}^{\lceil \log n \rceil} \left[2^{i+1} \log 2^i - \log 2^i - 2^i \log 2^i + 2^i \right] \tag{17}$$

$$T(n) = \frac{1}{2}\ln 2\left[2\log 2\sum_{i=1}^{\lceil \log n\rceil}\left[2^i i - \log 2\sum_{i=1}^{\lceil \log n\rceil} i - \log 2\sum_{i=1}^{\lceil \log n\rceil} 2^i i + \sum_{i=1}^{\lceil \log n\rceil} 2^i\right]\right] \tag{18}$$

Por otro lado.

$$\sum_{i=1}^{\lceil \log n\rceil} 2^i i = 2^1 1 + 2^2 2 + 2^3 3 + ... + 2^{\log n - 1}\log n - 1 + 2^{\log n}\log n \tag{19}$$

Se puede tomar $\log n$ como factor común y acotar por arriba a esta expresión de la siguiente manera.

$$\sum_{i=1}^{\lceil \log n\rceil} 2^i i \leq \log n[2^1 + 2^2 + 2^3 + ... + 2^{\log n - 1} + 2^{\log n}] = (n-1)\log n \tag{20}$$

Resolviendo la Ecuación 18.

$$T(n) \leq \frac{1}{2}\ln 2\left[2\log 2((n-1)\log n) - \log 2(\frac{\log^2 n + \log n}{2}) - \log 2((n-1)\log n) + (n-1)\right] \tag{21}$$

$$T(n) = O(n\log n) \tag{22}$$

Teorema 1. *Sea T un árbol binario balanceado de n hojas, donde no se privilegían los candidatos de R_2 ni R_3. El número de pasos promedio que tarda el árbol T para ser estable con respecto a R_1 es $O(n\log n)$.*

El Teorema 1 se puede interpretar como el hecho de que el máximo global se encuentra en la raíz, pero en la realidad el máximo global se puede encontrar en cualquier nodo diferente de la raíz. Si el máximo global es distinto a la raíz, los valores externivel de cada nodo son iguales al caso en el que la raíz es el máximo global. Los valores internivel son distintos para ambos árboles. Si se supone que cualquier nodo puede ser el máximo global, el número de pasos promedio que se necesita para estabilizar el árbol sigue siendo $O(n \log n)$ por que los límites de la sumatoria en la Ecuación 10 siguen siendo $O(\log n)$.

III.4 Estabilización con respecto a R_1 en un árbol de profundidad logarítmica doble

El árbol de profundidad logarítmica doble, en comparación con el árbol binario balanceado, tiene menor profundidad pero mayor número de nodos en cada externivel.

Las propiedades de este árbol son las siguientes:

1. Tiene n hojas, dónde $n = 2^{2^k}$ y k es un entero positivo.
2. La raíz tiene $2^{2^{k-1}} = n^{\frac{1}{2}}$ hijos.
3. En el internivel i, con respecto a la raíz, cada nodo tiene $2^{2^{k-i-1}}$ hijos.
4. El número de nodos en el internivel i, con respecto a la raíz, es $2^{2^k - 2^{k-i}} = n^{1-\frac{1}{2^i}}$, para $0 \leq i < k$.
5. Cada nodo en el internivel k, con respecto a la raíz, tiene 2 hijos.
6. El número de nodos en el internivel k, con respecto a la raíz, es $\frac{n}{2}$.

La Figura 13 muestra un ejemplo de un árbol de profundidad logarítmica doble. En este árbol existen n hojas, siguiendo las características del árbol que se enlistan en esta sección, se puede ver cómo en el internivel 1 (con respecto a la raíz) existen $n^{\frac{1}{2}}$ nodos y como en el internivel 2 existen $n^{\frac{3}{2^2}} = n^{\frac{3}{4}}$ nodos, de tal forma que en el internivel i existen $n^{1-\frac{1}{2^i}}$ nodos.

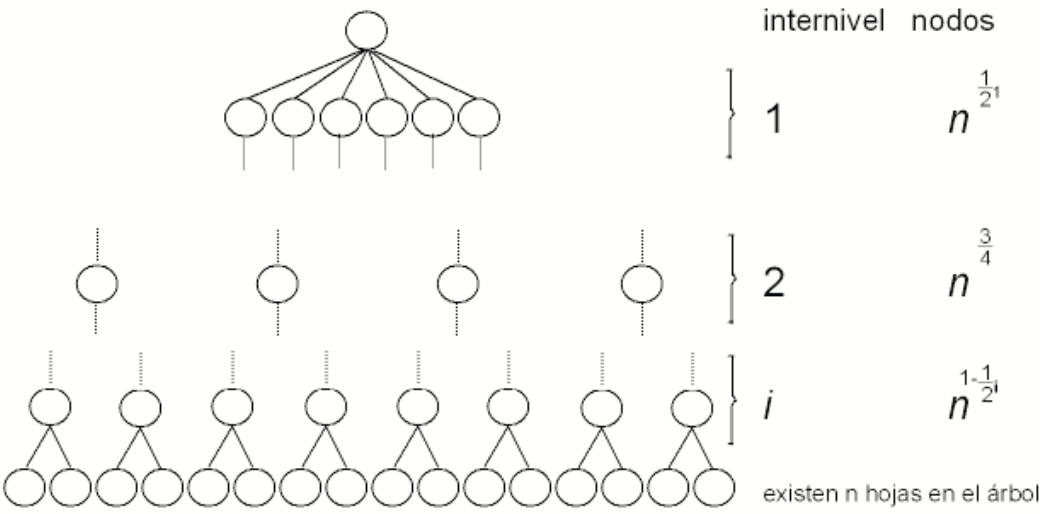

Figura 13: Árbol de profundidad logarítmica doble de n hojas

Suponga un árbol de profundidad logarítmica doble de n hojas y un árbol binario balanceado de n hojas también. Si los límites de la Ecuación 10 se ajustan al externivel 1 únicamente se ve que se necesitan $O(n \log n)$ pasos para estabilizar el externivel 1 en ambos árboles, por lo tanto se necesitan al menos $\Omega(n \log n)$ pasos para estabilizar el árbol de profundidad logarítmica doble.

Si se supone que tanto el árbol binario balanceado como el de profundidad logarítmica doble tienen el mismo número de hojas se observa que en ambos casos existen el mismo número de nodos en el externivel 1, sin embargo, el árbol binario balanceado tiene $\log n$ externiveles mientras que el de profunidad logarítmica doble tiene $\log \log n$ externiveles; esto significa que el árbol binario balanceado tiene más nodos en total, y que para cada externivel i, donde $1 < i \leq \log \log n$, el número de nodos del externivel i en el árbol de

profundidad logarítmica doble es menor al número de nodos del externivel i en el árbol binario balanceado. Adicionalmente, si el árbol binario balanceado de n hojas requiere en promedio $O(n \log n)$ pasos para ser estable (Teorema 1) teniendo éste más nodos que el de profundidad logarítmica doble, entonces la cota superior para el árbol de profundidad logarítmica doble es $O(n \log n)$ pasos. Como ambas cotas son la misma, se concluye el Corolario 1.

Corolario 1. *Sea T un árbol de profundidad logarítmica doble de n hojas donde no se privilegían los candidatos de R_2 ni R_3, el número de pasos promedio que el árbol T tarda para ser estable con respecto a R_1 es $O(n \log n)$.*

III.5 Estabilización con respecto a R_1 en un árbol arbitrario de diámetro $O(\log n)$

En este caso no se conoce la topología exacta del árbol, no se sabe cuántos nodos forman parte de cada externivel, ni tampoco se sabe cuántos vecinos tiene cada nodo. Lo único que se conoce es que el árbol tiene n nodos y que el diámetro es $O(\log n)$.

Sea n_i la cantidad de nodos en el externivel i, de tal manera que $\sum_{i=0}^{c\lceil \log n \rceil} n_i = n$, donde c es una constante. Si el diámetro del árbol es $c \log n$, donde c es una constante, y tiene n nodos, entonces el valor esperado de n_i es $E[n_i] = \frac{n}{c \log n}$. Esto significa que al menos existe un externivel con al menos $\frac{n}{c \log n}$ nodos y que el número de hojas n_0 es igual o mayor que $\frac{n}{c \log n}$. Sea p la probabilidad de que un nodo del árbol sea candidato para la regla R_1, para calcular el valor de z_i hay que multiplicar los $\frac{n}{c \log n}$ nodos que en promedio se encuentran en cada externivel i por el número de externiveles que hay entre i y $c \log n$, por tal razón $E[z_i] = \frac{n}{c \log n}(c \log n - i)p$ y $E[y_i] = \frac{n}{c \log n}p$. Sustituyendo en la Ecuación 10 se tiene que el número de pasos promedio $T(n)$ requeridos para estabilizar el árbol con respecto a R_1 es:

$$T(n) = \ln 2 \sum_{i=1}^{c\lceil \log n \rceil} \left[\left(\frac{n}{c \log n}(c \log n - i)p - \frac{n}{c \log n}p \right) \log \frac{n}{c \log n}p + \frac{n}{c \log n}p \right] \tag{23}$$

$$T(n) = \ln 2 \sum_{i=1}^{c\lceil \log n \rceil} \left[\left(\frac{n}{c \log n}ip - \frac{n}{c \log n}p \right) \log \frac{n}{c \log n}p + \frac{n}{c \log n}p \right] \tag{24}$$

$$T(n) = \ln 2 \sum_{i=1}^{c\lceil \log n \rceil} \left[\frac{np}{c \log n}(i-1) \log \frac{np}{c \log n} + \frac{np}{c \log n} \right] \tag{25}$$

$$T(n) = \frac{np}{c \log n} \ln 2 \sum_{i=1}^{c\lceil \log n \rceil} \left[(i-1) \log \frac{np}{c \log n} + 1 \right] \tag{26}$$

$$T(n) = \frac{np}{c \log n} \ln 2 \sum_{i=1}^{c\lceil \log n \rceil} \left[(i-1) \log \frac{np}{c \log n} + 1 \right] \tag{27}$$

$$T(n) = \frac{np}{c \log n} \ln 2 \left[\log \frac{np}{c \log n} \sum_{i=1}^{c\lceil \log n \rceil} (i-1) + \sum_{i=1}^{c\lceil \log n \rceil} 1 \right] \tag{28}$$

La sumatoria

$$\sum_{i=1}^{c\lceil \log n \rceil} i - 1 = O(\log^2 n) \tag{29}$$

por lo tanto

$$T(n) = O(n \log^2 n) \tag{30}$$

Teorema 2. *Sea T un árbol arbitrario de n nodos y diámetro $O(\log n)$ donde no se privilegían los candidatos de R_2 ni R_3. El número de pasos promedio que tarda T para ser estable con respecto a R_1 es $O(n \log^2 n)$.*

III.6 Estabilización de acuerdo a R_2

La regla R_2 dice que si el nodo i tiene exactamente un vecino j con un valor $x_j > x_i$ entonces $x_i = x_j - 1$, es decir, x_i se hace uno menos que el valor de su vecino mayor.

Nota: en el trabajo original (Xu y Srimani, 2005), los autores no aclaran explícitamente qué sucede cuando $x_i = x_j$, pero a través del análisis del algoritmo se concluye que el algoritmo funciona adecuadamente para $x_j \geq x_i$.

Se dice que un árbol es estable con respecto a R_2 si

1. Ningún nodo es candidato de R_2.
2. Existe un máximo global único de tal forma que, para todo nodo i, la diferencia $x_{max} - x_i$ es igual a la distancia que hay entre el nodo i y el máximo global.

Dado un árbol arbitrario $T = (V, E)$, el internivel de un nodo es 0 si éste es el máximo global, 1 para los vecinos del máximo global, y así el internivel va aumentando de uno en uno hasta llegar a las hojas del árbol.

Para estabilizar un árbol con respecto a R_2 la tendencia del algoritmo es iniciar la estabilización en los vecinos del máximo global y dirigirse hacia las hojas. El nodo con internivel 0 ya está estable con respecto a R_2 ya que no tiene vecinos mayores y por lo tanto no es candidato de R_2. Los nodos con internivel 1 se estabilizan con respecto a R_2 después de que todos los nodos candidatos con este internivel ejecuten las operaciones de estabilización de la regla R_2, esto sucede por que la regla R_2 no modifica el valor del máximo global y una vez que los vecinos de éste ajustan sus valores, ya nunca ejecutarán las operaciones de la regla R_2. De la misma manera, los nodos con internivel i sólo pueden ser estables hasta que todos los nodos con internivel menor que i lo sean.

Sea y_i la cantidad de candidatos en el internivel i y $z_i = \sum_{j=i}^{k} y_j$ el número de candidatos en todos los interniveles desde i hasta k (donde k es el internivel máximo). Para calcular el promedio de pasos necesarios para estabilizar un árbol con respecto a R_2, primero se deben calcular el número de pasos necesarios para estabilizar un internivel arbitrario i y se supone que los interniveles $j < i$ ya son estables.

En el árbol existen candidatos de R_2 en el internivel i y en cada internivel $j > i$. Considerando el calendarizador justo, éste puede privilegiar un nodo de i o un nodo de j con la misma probabilidad. La probabilidad para privilegiar algún nodo de i es por lo tanto $\frac{y_i}{z_i}$. Sea k_i una variable aleatoria que representa el número de pasos que se requieren para que un nodo candidato del internivel i se privilegíe. El valor esperado de k_i es $\frac{z_i}{y_i}$ (por tener k_i una distribución geométrica). Una vez que se privilegía algún nodo de i, el número de pasos esperados para privilegiar otro nodo del internivel i es $\frac{z_i-1}{y_i-1}$. El número de pasos esperados para privilegiar el j-ésimo candidato del internivel i es $\frac{z_i-j}{y_i-j}$. En total, el número de pasos T_i que se necesitan en promedio para privilegiar todos los candidatos del internivel i es:

$$T_i = \sum_{j=0}^{j=y_i-1} \frac{z_i - j}{y_i - j} \tag{31}$$

$$T_i = \sum_{j=0}^{j=y_i-1} \frac{z_i}{y_i - j} - \sum_{j=0}^{j=y_i-1} \frac{j}{y_i - j} \tag{32}$$

$$T_i = z_i \sum_{j=1}^{j=y_i} \frac{1}{j} - \sum_{j=1}^{j=y_i} \frac{y_i - j}{j} \tag{33}$$

$$T_i = (z_i - y_i) \sum_{j=1}^{y_i} \frac{1}{j} + \sum_{j=1}^{y_i} \frac{j}{j} \tag{34}$$

La sumatoria $H_n = \sum_{j=1}^{n} \frac{1}{j}$ se conoce como el n-ésimo número armónico y su valor es $\ln n + O(1)$. Por lo tanto.

$$T_i = (z_i - y_i)(\ln y_i + O(1)) + y_i \tag{35}$$

Para una n suficientemente grande, $\ln y_i = o(1)$, entonces la Ecuacion 35 se puede simplificar a (36).

$$T_i = (z_i - y_i) \ln y_i + y_i \tag{36}$$

En promedio, el número de esperado de pasos $T(n)$ para estabilizar con respecto a R_2 a cualquier árbol de n nodos y externivel máximo k está dado por:

$$T(n) = \sum_{i=1}^{k} T_i = \sum_{i=1}^{k} (z_i - y_i) \ln y_i + y_i \tag{37}$$

La Ecuación 37 se puede representar como logaritmo de base dos. Si $\log y_i = \frac{\ln y_i}{\ln 2}$, entonces $\ln y_i = \ln 2 \log y_i$, por lo tanto $T(n)$ es igual a.

$$T(n) = \ln 2 \sum_{i=1}^{k} (z_i - y_i) \log y_i + y_i \tag{38}$$

Note que la Ecuación 38 es igual a la Ecuación 10, pero como se explicó en esta sección, los valores z_i y y_i son candidatos de R_2 y el índice i representa el internivel.

III.7 Estabilización con respecto a R_2 en un árbol binario balanceado

Suponga un árbol binario balanceado de n hojas. Suponga que el árbol es estable con respecto a R_1, como existe un máximo global único, todos los nodos del árbol son

candidatos de R_2. Suponga que el máximo global se encuentra en el centro del árbol de tal manera que existen $\log n$ interniveles.

Con esta información se definen los valores de z_i y y_i. El valor esperado de z_i es el número total de nodos en el árbol $(2n-1)$ menos el número de nodos que existen desde el internivel 0 al $i-1$, esto es 2^i-1.

$$E[z_i] = 2n - 1(2^i - 1) = 2n - 2^i \tag{39}$$

El valor esperado de y_i es

$$E[y_i] = 2^i \tag{40}$$

Usando la Ecuación 10 se puede calcular el número pasos promedio $T(n)$ necesarios para estabilizar el árbol binario balanceado de n hojas con respecto a R_2.

$$T(n) = \ln 2 \sum_{i=1}^{\lceil \log n \rceil} \left[\left(2n - 2^i - 2^i\right) \log 2^i + 2^i \right] \tag{41}$$

$$T(n) = \ln 2 \sum_{i=1}^{\lceil \log n \rceil} \left[\left(2n - 2^{i+1}\right) i + 2^i \right] \tag{42}$$

$$T(n) = \ln 2 \sum_{i=1}^{\lceil \log n \rceil} \left[2ni - 2^{i+1} i + 2^i \right] \tag{43}$$

$$T(n) \leq \ln 2 \left[2n \left(\frac{\log^2 n + \log n}{2} \right) - n \log n + n \right] \tag{44}$$

Teorema 3. *Sea T un árbol binario balanceado de n hojas que es estable con respecto a R_1 y donde no se privilegían candidatos de R_3. El número de pasos promedio que tarda el árbol T para ser estable con respecto a R_2 es $O(n \log^2 n)$.*

III.8 Estabilización con respecto a R_2 en un árbol de profundidad logarítmica doble

Suponga un árbol de profundidad logarítmica doble de n hojas y un árbol binario balanceado de n hojas también, ambos estables con respecto a R_1.

Si se supone que tanto el árbol binario balanceado como el de profundidad logarítmica doble tienen el mismo número de hojas se observa que el árbol binario balanceado tiene $\log n$ interniveles mientras que el de profunidad logarítmica doble tiene $\log \log n$ (suponiendo que el máximo global es la raíz) y que el árbol binario balanceado tiene más nodos en total. Además, para cada internivel i, $1 < i \leq \log \log n$, el número total de nodos candidatos en el árbol de profundidad logarítmica doble es menor al número total de nodos candidatos en el árbol binario balanceado. Recuerde que el árbol binario balanceado de n hojas requiere en promedio $O(n \log^2 n)$ pasos para ser estable (Teorema 3) y que éste tiene más nodos que el de profundidad logarítmica doble, por tal razón este valor es una cota superior para el árbol de profundidad logarítmica doble.

Corolario 2. *Sea T un árbol de profundidad logarítmica doble de n hojas, estable con respecto a R_1, donde no se privilegían los candidatos de R_3, el número de pasos promedio que el árbol T tarda para ser estable con respecto a R_2 es $O(n \log^2 n)$.*

III.9 Estabilización con respecto a R_2 en un árbol arbitrario de diámetro $O(\log n)$

En este caso no se conoce la topología exacta del árbol, no se sabe cuántos nodos forman parte de cada internivel, ni tampoco se sabe cuántos vecinos tiene cada nodo. Lo único que se conoce es que el árbol tiene n nodos y que el diámetro es $O(\log n)$.

Sea n_i la cantidad de nodos en el internivel i, de tal manera que $\sum_{i=0}^{c\lceil \log n \rceil} n_i = n$, donde c es una constante. Si el diámetro del árbol es $c \log n$ y tiene n nodos, entonces el valor esperado de n_i es $E[n_i] = \frac{n}{c \log n}$. Esto significa que al menos existe un externivel con $\frac{n}{c \log n}$ nodos y $n_0 \geq \frac{n}{c \log n}$; esto es, el número de hojas es igual o mayor a $\frac{n}{c \log n}$. Sea p la probabilidad de que un nodo del árbol sea candidato para la regla R_2, para calcular el valor de z_i hay que multiplicar los $\frac{n}{c \log n}$ nodos que en promedio se encuentran en cada internivel por el número de interniveles que hay entre i y $c \log n$ (aunque el número de interniveles no sea igual al diámetro del árbol se puede suponer que la diferencia entre ambos valores es a lo más un factor de 2 y se incluye en la constante c del diámetro $c \log n$), por tal razón $E[z_i] = \frac{n}{c \log n}(c \log n - i)p$ y $E[y_i] = \frac{n}{c \log n}p$. Sustituyendo en la Ecuación 10 se tiene que el número de pasos promedio $T(n)$ requeridos para estabilizar el árbol con respecto a R_2 es:

$$T(n) = \ln 2 \sum_{i=0}^{c\lceil \log n \rceil - 1} \left[\left(\frac{n}{c \log n}(c \log n - i)p - \frac{n}{c \log n}p \right) \log \frac{n}{c \log n}p + \frac{n}{c \log n}p \right] \tag{45}$$

$$T(n) = \ln 2 \sum_{i=0}^{c\lceil \log n \rceil - 1} \left[\left(\frac{n}{c \log n}ip - \frac{n}{c \log n}p \right) \log \frac{n}{c \log n}p + \frac{n}{c \log n}p \right] \tag{46}$$

$$T(n) = \ln 2 \sum_{i=0}^{c\lceil \log n \rceil - 1} \left[\frac{np}{c \log n}(i-1) \log \frac{np}{c \log n} + \frac{np}{c \log n} \right] \tag{47}$$

$$T(n) = \frac{np}{c \log n} \ln 2 \sum_{i=0}^{c\lceil \log n \rceil - 1} \left[(i-1) \log \frac{np}{c \log n} + 1 \right] \tag{48}$$

$$T(n) = \frac{np}{c \log n} \ln 2 \sum_{i=1}^{c\lceil \log n \rceil} \left[(i-1-1) \log \frac{np}{c \log n} + 1 \right] \tag{49}$$

$$T(n) = \frac{np}{c \log n} \ln 2 \left[\log \frac{np}{c \log n} \sum_{i=1}^{c\lceil \log n \rceil} (i-2) + \sum_{i=1}^{c\lceil \log n \rceil} 1 \right] \tag{50}$$

La sumatoria

$$\sum_{i=1}^{c\lceil \log n \rceil} i - 2 = O(\log^2 n) \tag{51}$$

por lo tanto

$$T(n) = O(n \log^2 n) \tag{52}$$

Teorema 4. *Sea T un árbol arbitrario de n nodos y diámetro $O(\log n)$ que es estable con respecto a R_1 y donde no se privilegían candidatos de R_3. El número de pasos promedio que tarda T para ser estable con respecto a R_2 es $O(n \log^2 n)$.*

III.10 Combinación de las reglas R_1 y R_2 en un árbol binario balanceado

Calcular el tiempo de ejecución para ambas reglas por separado no es representativo de lo que realmente sucede durante la ejecución del algoritmo.

El tiempo de ejecución del algoritmo depende de la regla R_1, ya que ésta es la que establece el valor del máximo global en el árbol; es importante que el algoritmo primero termine de ejecutar las operaciones de la regla R_1 para estabilizar el árbol.

Mientras R_1 está en proceso de establecer el máximo global, la regla R_2 ajusta los valores a éste máximo global. Las ejecuciones de las operaciones de la regla R_1 pueden generar candidatos adicionales de la regla R_2. Entre dos ejecuciones de las operaciones de R_1 dentro de un externivel i pueden existir varias ejecuciones de las operaciones de la regla R_1 en otros externiveles, también pueden existir ejecuciones de las operaciones de la regla R_2 en todo el árbol (incluso entre los mismos nodos del externivel i).

En cualquier momento pueden existir nodos del conjunto C_1 tanto como nodos del conjunto C_2, y el calendarizador justo puede decidir privilegiar uno u otro sin importar cuántos hay de cada uno o cómo ese movimiento cambia el estado de otros nodos. Es necesario calcular cuántos pasos se requieren para terminar de ejecutar las operaciones de la regla R_1 si entre dos ejecuciones de las operaciones de la regla R_1 por nodos del externivel i puede haber una o más ejecuciones de las operaciones de la regla R_2. Por tal razón, se supone que la probabilidad de privilegiar un candidato de R_1 es igual a la probabilidad de privilegiar un candidato de R_2, y que a través de cada iteración el valor de $z_{\log n} = E[|C_1|] + E[|C_2|]$ se mantiene constante, o sea, el número de candidatos en todo el árbol no disminuye. Si $z_{\log n}$ es una constante y y_i es el número de candidatos de R_1 en el externivel i, la probabilidad de que uno de estos candidatos se privilegíe es $\frac{y_i}{z_{\log n}}$ mientras que para el j-ésimo candidato de i es $\frac{y_i - j}{z_{\log n}}$. Sea k_i una variable aleatoria que representa el número de pasos que se requieren para que un nodo candidato de i se privilegíe. El valor esperado de k_i es $\frac{z_{\log n}}{y_i}$ (por tener k_i una distribución geométrica). El número de pasos esperados para privilegiar el j-ésimo candidato de i es $\frac{z_{\log n}}{y_i - j}$. El número de pasos promedio T_i requeridos para privilegiar todos los nodos del externivel i es

$$T_i = \sum_{j=1}^{j=y_i-1} \frac{z_{\log n}}{y_i - j} \tag{53}$$

$$T_i = z_{\log n} \sum_{j=1}^{j=y_i-1} \frac{1}{y_i - j} \tag{54}$$

$$T_i = z_{\log n} \sum_{j=1}^{j=y_i-1} \frac{1}{j} \tag{55}$$

$$T_i = z_{\log n} \ln y_i \tag{56}$$

Se multiplica $\ln y_i$ por una constante para cambiar el logaritmo natural a base dos y se resuelve la siguiente sumatoria para calcular el número de pasos promedio $T(n)$ para estabilizar el árbol con respecto a R_1.

$$T(n) = \sum_{i=1}^{\lceil \log n \rceil} T_i = \sum_{i=1}^{\lceil \log n \rceil} z_{\log n} \log y_i \tag{57}$$

$$T(n) = z_{\log n} \sum_{i=0}^{\lceil \log n \rceil - 1} \log y_i \tag{58}$$

$$T(n) = z_{\log n} \sum_{i=1}^{\lceil \log n \rceil} \log \frac{2^{i-1}}{2} \tag{59}$$

$$T(n) = z_{\log n} \sum_{i=1}^{\lceil \log n \rceil} \log 2^{i-2} \tag{60}$$

$$T(n) = z_{\log n} \log 2 \left[\sum_{i=1}^{\lceil \log n \rceil} i - \sum_{i=1}^{\lceil \log n \rceil} 2 \right] \tag{61}$$

La sumatoria

$$\sum_{i=1}^{\lceil \log n \rceil} i \tag{62}$$

es igual a

$$\frac{\log n(\log n + 1)}{2} \tag{63}$$

por lo tanto

$$T(n) = z_{\log n} \log 2 \left[\frac{\log^2 n + \log n}{2} - 2 \log n \right] \tag{64}$$

Una vez que el árbol se estabiliza con respecto a R_1 es necesario que se estabilice con respecto a R_2. Ya se comprobó en el Teorema 3 que para estabilizar un árbol binario balanceado con respecto a R_2 se necesitan en promedio $O(n \log n)$ pasos. Así se establece el Teorema 5.

Teorema 5. *Sea T un árbol binario balanceado de n hojas, y sea $z_{\log n} = O(n)$. El número de pasos promedio que tarda T para ser estable es $O(n \log^2 n)$.*

III.11 Combinación de las reglas R_1 y R_2 en un árbol de profundidad logarítmica doble

Suponga un árbol de profundidad logarítmica doble de n hojas y un árbol binario balanceado de n hojas también. Al igual como en los casos anteriores, si se supone que ambos tienen el mismo número de hojas se observa que el árbol binario balanceado tiene $\log n$ externiveles mientras que el de profunidad logarítmica doble tiene $\log \log n$; esto significa que el árbol binario balanceado tiene más nodos en total, y que para

cada externivel i, $1 < i \leq \log\log n$, el número de nodos del externivel i en el árbol de profundidad logarítmica doble es menor al número de nodos del internivel i en el árbol binario balanceado. Adicionalmente, si el árbol binario balanceado de n hojas requiere en promedio $O(n \log^2 n)$ pasos para ser estable (Teorema 4) teniendo éste más nodos que el de profundidad logarítmica doble, entonces una cota superior para el árbol de profundidad logarítmica doble es $O(n \log^2 n)$ pasos. Así, se establece el Corolario 3.

Corolario 3. *Sea T un árbol de profundidad logarítmica doble de n hojas, el número de pasos promedio que el árbol T tarda para ser estable es $O(n \log^2 n)$.*

III.12 Probabilidad de terminación del algoritmo en un árbol binario balanceado

Ya se demostró que el algoritmo presentado en (Xu y Srimani, 2005) utiliza en promedio $O(n \log^2 n)$ pasos cuando se aplica un calendarizador justo en un árbol binario balanceado. Ahora se demuestra que la probabilidad de que el algoritmo termine utilizando más de $O(n \log^2 n)$ pasos es mínima.

Para hacer este cálculo, y facilitar la explicación, se utiliza el problema del coleccionista de cupones descrito por Motwani y Raghavan en (Motwani y Raghavan, 1995). En este problema existen n tipos de cupones diferentes y en cada intento se selecciona un cupon al azar. Cada cupón tiene la misma probabilidad de ser de cualquiera de los n tipos, y la selección de cada cupón es independiente en cada intento. Si m es el número de intentos, la meta del problema es estudiar la probabilidad de haber tomado al menos un cupón de cada tipo después de m intentos.

Motwani y Raghavan (1995) establecen el siguiente teorema:

Teorema 6. *Sea X la variable aleatoria que denota el número de intentos para tomar al menos un cupón de cada uno de los n tipos de cupones diferentes. Entonces para cualquier $c \in \Re$ y $m = n \ln n + cn$, la probabilidad de que $X > m$ es $Pr[X > m] = 1 - e^{-e^{-c}}$.*

En otras palabras, la probabilidad de que se necesiten más de $n \ln n + cn$ intentos para obtener al menos un cupón de cada tipo es $1 - e^{-e^{-c}}$. Por otro lado, la probabilidad de que se necesiten cuando mucho $n \ln n + cn$ intentos es $e^{-e^{-c}}$.

El proceso de estabilización de un árbol se puede modelar como una secuencia de problemas del coleccionista de cupones. De hecho, el proceso de estabilización de cada externivel del árbol se puede modelar como un problema del coleccionista de cupones diferente. El número de cupones diferentes corresponden al número de nodos candidatos que se podrían privilegiar para estabilizar un externivel i (esto incluye los candidatos del externivel i y de los externiveles superiores). De la misma manera, el número de intentos para obtener todos los cupones corresponde a los pasos que se necesitan en promedio para estabilizar todo el externivel i.

Se sabe por la Ecuación 8 que se necesitan en promedio $(z_i - y_i) \ln y_i + y_i$ pasos para estabilizar cualquier externivel o internivel con respecto a cualquier regla. Un árbol se debe estabilizar primero con respecto a R_1 y después con respecto a R_2, pero mientras que el árbol se estabiliza con respecto a R_1, los candidatos de R_2 también se pueden privilegiar. El valor de z_i para cualquier externivel i es cuando mucho n, por esta razón el número de pasos necesarios para estabilizar a i se puede acotar por arriba a $n \ln n + n$, por lo tanto, el número de pasos necesarios $T(n)$ para estabilizar un árbol binario balanceado con respecto R_1 es,

$$T(n) = (n \ln n + n) \log n = n \ln 2 \log^2 n + n \log n \tag{65}$$

El número de pasos necesarios para estabiliza un árbol con respecto a R_2 no es mayor

al número de pasos necesarios para estabilizarlo con respecto a R_1. Note que el orden de este valor es igual al valor que se calculó en la sección 10 de este capítulo.

El evento E_i, para $1 \leq i \leq \log n$, se define como el hecho de estabilizar al externivel i utilizando más de $n \ln n + cn$ pasos, por el Teorema 6 se sabe que la probabilidad de que el evento E_i ocurra es $1 - e^{-e^{-c}}$. De la misma manera, se define al evento E_i' como el hecho de estabilizar al externivel i utilizando cuando mucho $n \ln n + cn$ pasos, donde la probabilidad de que E_i' ocurra es $e^{-e^{-c}}$.

Cada evento E_i' es independiente, es decir, el hecho de que E_i' ocurra no influye en que ocurra o no algún evento E_j', para $j > i$. Para que todo el árbol se estabilice dentro del tiempo definido, cada evento E_i' debe ocurrir, es decir, que $E_1' \cap E_2' \cap E_3' \cap ... \cap E_{\log n}'$ ocurra. Por lo tanto, la probabilidad P de que ésto suceda es.

$$P = (e^{-e^{-c}})^{\log n} \tag{66}$$

$$P = e^{-e^{-c} \log n} \tag{67}$$

$$P = e^{\log n - e^{-c}} \tag{68}$$

$$P = e^{\frac{\ln n - e^{-c}}{\ln 2}} \tag{69}$$

$$P = e^{\ln n \frac{-e^{-c}}{\ln 2}} \tag{70}$$

$$P = n^{\frac{-e^{-c}}{\ln 2}} \tag{71}$$

De aquí se concluye el siguiente teorema.

Teorema 7. *Sea T un árbol binario balanceado de n nodos. Con una probabilidad de $n^{\frac{-e^{-c}}{\ln 2}}$, T se estabiliza en menos de $n \ln 2 \log^2 n + cn \log n$.*

Ejemplo 4: se tiene un árbol binario balanceado de 1000 nodos, y se define que $c = 7$, de tal manera que $T(n) = 1000 \ln 2 \log^2 1000 + 7(1000) \log 1000$. La probabilidad de que el árbol se estabilice utilizando cuando mucho $T(n)$ pasos es $1000^{\frac{-e^{-7}}{\ln 2}} = 0.99$. Note que la probabilidad de que el algoritmo termine utilizando un número de pasos del mismo orden que el promedio es bastante alta.

Capítulo IV

Análisis del algoritmo en un árbol generado aleatoriamente

En el capítulo III se calcula el tiempo promedio de ejecución del algoritmo auto-estabilizante de Xu y Srimani (Xu y Srimani, 2005) en árboles específicos. En el presente capítulo se calcula el tiempo de ejecución esperado en árboles generados aleatoriamente. Para hacer este cálculo, es importante definir el método para generar el árbol aleatorio, una vez que se define dicho método se puede calcular el diámetro esperado del árbol, y así aplicar la Ecuación 10.

En este capítulo se proponen cuatro métodos diferentes de generar un árbol aleatorio y el análisis hecho para calcular el diámetro esperado de cada árbol.

IV.1 Primer método

El primer método para generar el árbol aleatorio se modela de manera similar al problema de ocupación (Motwani y Raghavan, 1995). En un problema de ocupación se tienen m objetos indistinguibles entre sí que se deben asignar a n distintas clases. Motwani y Raghavan explican el problema como el hecho de tener m pelotas y arrojarlas a n cajas distintas, y ver cuál es la distribución de las pelotas sobre todas las cajas. Este método en particular genera árboles con una topología similar a una estrella enraizada, además se supone que las ramas del árbol no tiene bifurcaciones.

IV.1.1 Suposiciones

Se supone que se tienen n nodos, donde uno de ellos es la raíz del árbol y cada uno de los otros $n-1$ nodos deben conectarse aleatoriamente en alguna rama del árbol. En este método, las pelotas representan los nodos y cada una de las cajas representa una rama que posiblemente puede tener el árbol (una rama es un subárbol de uno o más nodos conectado a la raíz). Asi como en el problema de ocupación, se supone que una pelota puede caer en cualquier caja con la misma probabilidad, sin importar si está vacia o ocupada; de esta manera, la probabilidad de que un nodo se conecte a cualquier rama es $\frac{1}{n-1}$ (la probabilidad de que la pelota caiga en alguna caja es independiente de lo que ocurra en intentos anteriores). Adicionalmente, cada rama del árbol se forma como una cadena de nodos.

IV.1.2 Descripción del método

Este método es sencillo si se tiene en mente el problema de ocupación: se tienen $n-1$ cajas vacías y aleatoriamente se deben colocar $n-1$ pelotas dentro de estas cajas. Este método es similar al problema de ocupación porque cada caja representa una posible rama y el número de pelotas en la caja representa el número de nodos en la rama respectiva, de tal forma que al finalizar el método las dos cajas con mayor número de pelotas representa el diámetro del árbol.

La Figura 14 muestra un ejemplo sencillo de este metodo. Se tienen 7 nodos, de los cuáles uno de ellos es la raíz y los otros 6 se conectan aleatoriamente al árbol. En la Figura 14a se muestra el nodo raíz y las 6 posibles ramas que pueden existir en el árbol. En la Figura 14b se muestra cómo los 6 nodos se conectan aleatoriamente a alguna de las 6 posibles ramas del árbol. La Figura 14c muestra el árbol que se genera con este método considerando que cada rama es una cadena de nodos.

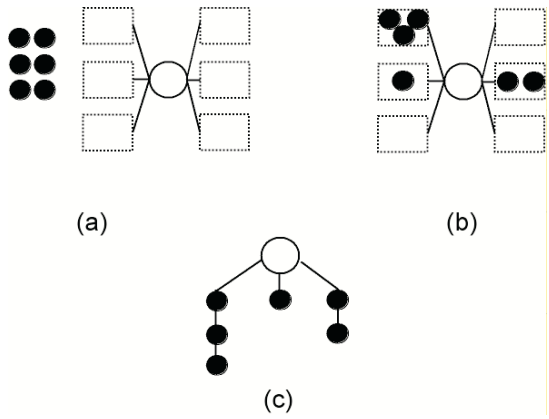

Figura 14: Árbol generado aleatoriamente con el primer método. La figura a muestra 6 pelotas que se deben arrojar en 6 cajas. La figura b muestra las pelotas dentro de las cajas. La figura c muestra el árbol equivalente

IV.1.3 Análisis

Motwani y Raghavan demuestran que si existen n cajas y $n \log n$ pelotas, con probabilidad de al menos $1 - \frac{1}{n}$ cada caja contiene $O(\log n)$ pelotas, es decir, si se tienen $n \log n$ nodos, al utilizar este método el diámetro del árbol es $O(\log n)$. Por lo tanto, $O(\log n)$ es una cota superior para el diámetro de este árbol.

Para calcuar el número de ramas en el árbol se calcula cuántas cajas ocupadas existen al finilizar el procedimiento. Sea la variable indicadora $X_i = 1$ si la caja i tiene al menos una pelota y $X_i = 0$ si está vacía. La probabilidad de que la pelota 1 caiga en la caja i es $\frac{1}{n-1}$ (ya que la primer pelota se considera la raíz del árbol). La probabilidad de que la pelota 2 caiga la caja i también es $\frac{1}{n-1}$. Para cada pelota j, $1 \leq j \leq n$, la probabilidad de que la pelota j caiga en la caja i es $\frac{1}{n-1}$. Por lo tanto, la probabilidad de que alguna de las $n-1$ pelotas caiga en la caja i es $\frac{1}{n-1} \times n - 1 = 1$. El número esperado de cajas ocupadas C está dado por

$$C = \sum_{i=1}^{n-1} X_i = O(n) \tag{72}$$

Con el primer método se generan árboles aleatorios similares a una estrella enraizada, donde el número de ramas es $O(n)$ y la profundidad de cada rama es a lo más $O(\log n)$

con alta probabilidad.

IV.2 Segundo método

El segundo método es similar al primero, en lo único que difieren es en la probabilidad que tienen los nodos para conectarse al árbol. En el primer método la probabilidad siempre es la misma en cada iteración, pero en el segundo método la probabilidad cambia conforme se van agregando nodos al árbol. Al igual que el método 1, las ramas del árbol que se forman con este método son cadenas de nodos (sin bifurcaciones).

IV.2.1 Suposiciones

Existe un nodo raíz donde el primer nodo $i = 1$ se conecta forzosamente a la raíz y cada nodo $i > 1$ se conecta a la raíz o a cualquier hoja con la misma probabilidad. El hecho de que un nodo se conecte al árbol es independiente de lo que ocurra con otros nodos.

IV.2.2 Descripción del método

El nodo $i = 1$ se conecta a la raíz. En cada nodo i, donde $1 < i \leq n - 1$, si el árbol tiene m hojas antes de conectar el nodo i, entonces la probabilidad de que el nodo se conecte a una hoja o a la raíz es $\frac{1}{m+1}$.

IV.2.3 Análisis

Para este método primero se calculan cuántas ramas se tienen al finalizar el proceso. Una vez que se sabe cuántas ramas tiene el árbol se debe calcular el número máximo de nodos en las dos ramas más grandes (la suma de sus tamaños es el diámetro del árbol).

Si inicialmente se tiene un nodo conectado a la raíz, de tal manera que sólo existe una rama, sea X_i una variable aleatoria indicadora donde $X_i = 1$ cuando hubo un éxito, es decir, el nodo i se conecta a la raíz (formando así una rama nueva) y $X_i = 0$ cuando suceda lo contrario. La probabilidad de conectar el nodo $i = 2$ a la raíz es $\frac{1}{2}$. La probabilidad de conectar cualquier nodo a la raíz es $\frac{1}{k}$, donde k es la suma del número de ramas más la raíz. Por tener X_i una distribución geométrica, el número esperado de pasos antes de obtener un éxito es el inverso de la probabilidad de éxito, es decir, si se tienen r ramas y una raíz, de tal manera que $r + 1 = k$, se necesitan k pasos antes de generar una rama nueva.

Se divide el procedimiento en épocas, donde la época i se define como el instante cuando el árbol tiene i ramas. Se sabe que se ocupan k pasos para cambiar de época, sin embargo se debe calcular el número de épocas x que existen después de conectar los $n - 1$ nodos al árbol. Esto se resuelve con la siguiente ecuación.

$$\sum_{k=1}^{x} k = n - 1 \tag{73}$$

$$\frac{x(x+1)}{2} = n - 1 \tag{74}$$

$$x^2 + x = 2n - 2 \tag{75}$$

Resolviendo la Ecuación cuadrática 75 se obtiene que el número esperado de ramas en el árbol es $x = O(n^{\frac{1}{2}})$. Se puede hacer un análisis similar para calcular cuántos nodos se conectan a las dos primeras ramas (ramas más antiguas), en ambos casos es $O(n^{\frac{1}{2}})$. Por lo tanto el diámetro esperado del árbol es,

$$\sum_{i=1}^{x} i = n - 2 \tag{76}$$

$$\frac{x(x+1)}{2} = n - 2 \tag{77}$$

$$x^2 + x = 2n - 4 \tag{78}$$

$$x = O(n^{\frac{1}{2}}) \tag{79}$$

Este método genera árboles similares a los que se generan con el primer método, pero el diámetro del árbol es mayor en este caso.

IV.3 Tercer método

El tercer método es una variante del segundo que genera más bifurcaciones en cada rama y un diámetro más pequeño (las ramas de este árbol no son necesariamente cadenas de nodos como en los dos primeros casos). Se puede ver como una versión recursiva del segundo método.

IV.3.1 Suposiciones

Al igual que en el segundo método, se supone que existe un nodo raíz y el primer nodo $i = 1$ se conecta forzosamente a la raíz. Cada nodo $i > 1$ se conecta a la raíz o a cualquier rama con la misma probabilidad.

IV.3.2 Descripción

Cada nodo nuevo puede conectarse a una rama o a la raíz del árbol (como en el segundo método), si el nodo se conecta a una rama, esta rama se considera un subárbol de tal manera que el nodo nuevo puede conectarse a la raíz de este subárbol o a una rama de este subárbol (como en el segundo método), y así sucesivamente hasta conectar todos los nodos. Las ramas de este árbol no son necesariamente cadenas de nodos.

En la Figura 15 se muestra un ejemplo del tercer método. En la Figura 15a cada nodo se conecta aleatoriamente a la raíz o a alguna de las k ramas del árbol como en el segundo método. Si se considera que cada rama del árbol completo es un subárbol, se pueden tomar todos los nodos de estos subárboles y aplicarles el segundo método otra vez. La Figura 15b muestra como se toman los nodos de cada rama y el resultado de volverles a aplicar el segundo método en cada subárbol. En la Figura 15c se les aplica de nuevo el segundo método a los nuevos subárboles para formar subárboles más pequeños. Se aplica el segundo método a cada subárbol hasta que el subárbol más pequeño solo tenga a lo más dos nodos.

IV.3.3 Análisis

Sea T el árbol generado usando el segundo método. Se observa que el número esperado de ramas en T es de $O(n^{\frac{1}{2}})$ y que su rama más grande tiene $O(n^{\frac{1}{2}})$ nodos (éste sería el diámetro en el segundo método, pero con el tercer método se espera que el diámetro sea menor ya que no es una cadena).

Sea T_1 la rama con más nodos del árbol T, al aplicarsele al subárbol T_1 el segundo método, se espera que T_1 tenga $O((n^{\frac{1}{2}})^{\frac{1}{2}}) = O(n^{\frac{1}{4}})$ ramas y su rama más grande tenga $O(n^{\frac{1}{4}})$ nodos. De la misma manera, sea T_2 la rama más grande de T_1 y se le aplica el

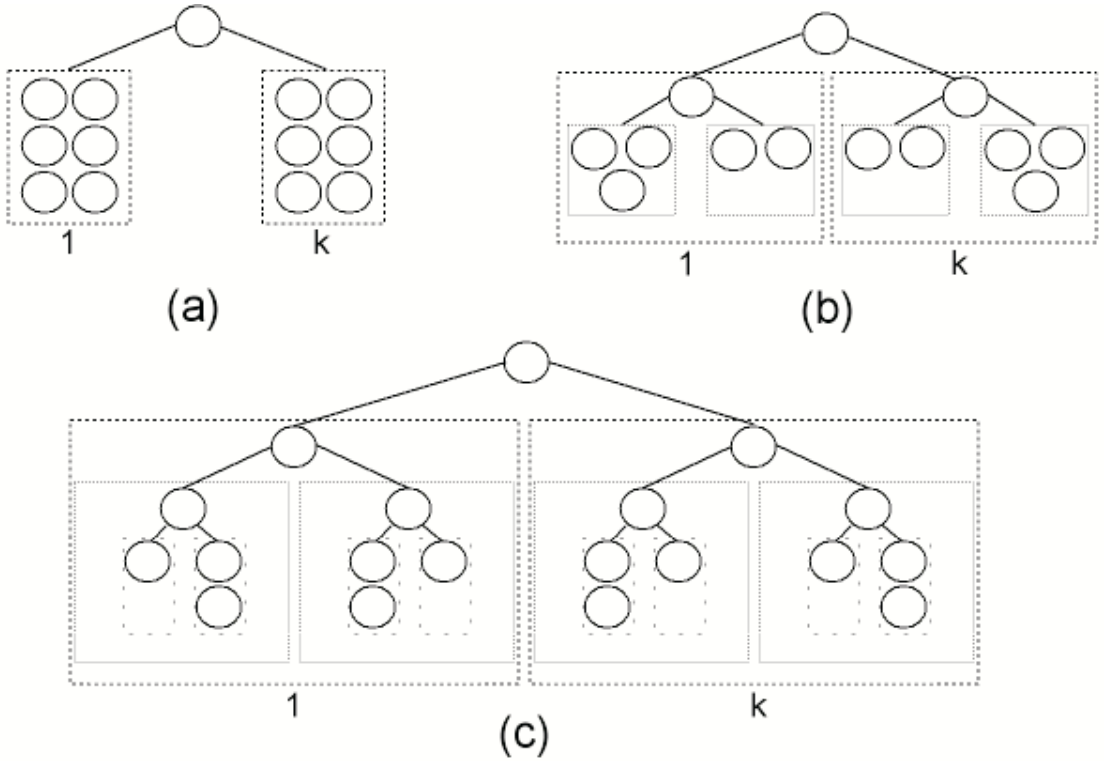

Figura 15: Árbol generado aleatoriamente con el tercer método. En la figura a cada nodo se conecta aleatoriamente a una rama. En la figura b a cada rama se le aplica el segundo método para formar subárboles. La figura c muestra como se sigue aplicando el segundo método hasta tener subárboles de a lo más dos nodos.

mismo método, el número esperado de ramas del subárbol nuevo es $O(n^{\frac{1}{8}})$ donde su rama más grande tiene $O(n^{\frac{1}{8}})$ nodos. Al continuar con este procedimiento el último subárbol (el más pequeño) tiene $n^{\frac{1}{2^k}} = 2$ nodos. Si se calcula el valor de k, se puede conocer el diámetro de todo el árbol. El valor de k se puede despejar de la siguiente ecuación.

$$n^{\frac{1}{2^k}} = 2 \tag{80}$$

$$\frac{1}{2^k} \log n = \log 2 \tag{81}$$

$$2^k = \log n \tag{82}$$

$$\log 2^k = \log \log n \tag{83}$$

$$k = \log \log n \tag{84}$$

El diámetro esperado del árbol utilizando este método es $O(\log \log n)$, en otras palabras, es de profundidad logarítmica doble.

Con este método se pueden generar árboles con diámetro muy pequeño y muchas bifurcaciones.

IV.4 Cuarto método

A diferencia de los otros tres métodos, este método no conecta nodos a las ramas sino a cualquier nodo que ya forma parte del árbol.

IV.4.1 Suposiciones

Se supone que el árbol consta de n nodos, un nodo i tiene la misma probabilidad de conectarse a cualquier nodo que ya forma parte del árbol. El nodo i se puede conectar a cualquiera de los $i-1$ nodos previos con probabilidad $\frac{1}{i-1}$. El hecho de que un nodo se conecte al árbol es independiente de lo que ocurra con otros nodos.

IV.4.2 Descripción del método

Este método al igual que los demás, inicia con un nodo raíz. Si j es el número de nodos que ya existen en el árbol entonces el i-ésimo nodo tiene una probabilidad de $\frac{1}{j}$ para conectarse

a cualquiera de los otros nodos, de tal manera que el primer nodo tiene probabilidad de 1 para conectarse a la raíz, el segundo tiene probabilidad de $\frac{1}{2}$ para conectarse a cualquiera de los dos ya presentes y así sucesivamente, hasta que todos los nodos se hayan conectado al árbol.

IV.4.3 Análisis

Para calcular el diámetro esperado del árbol se calcula en promedio cuánto crece la primera rama. Se supone que en la primera rama siempre existe un nodo de tal manera que si el i-ésimo nodo se conecta a éste, el diámetro del árbol crece. En cada iteración, la probabilidad de que el nodo nuevo se conecte a la primera rama es $\frac{1}{j}$ donde j es el número de nodos que ya existen en el árbol. Sea X_i una variable aleatoria indicadora que toma el valor de $X_i = 1$ cada vez que la rama más antigua crece en uno y $X_i = 0$ de otro modo. Entonces la longitud esperada L de la rama más vieja está dada por,

$$L = \sum_{i=1}^{n-1} \frac{1}{i} X_i \tag{85}$$

La ecuación se puede simplificar a,

$$L = \sum_{i=1}^{n-1} \frac{1}{i} \tag{86}$$

$$L = \sum_{i=1}^{n-1} \frac{1}{i} = \ln |n-1| + O(1) \tag{87}$$

El logaritmo natural resultante se puede representar en base dos. Como la longitud de la rama más grande es $O(\log n)$, en consiguiente el diámetro esperado de este árbol aleatorio es $O(\log n)$.

Sea X_i la variable aleatoria indicadora que toma el valor $X_i = 1$ cuando el nodo i se conecta a la raíz. La probabilidad de que el nodo i se conecte a la raíz siempre es $\frac{1}{i-1}$. El número esperado de ramas R del árbol es,

$$R = \sum_{i=1}^{n-1} \frac{1}{i-1} = O(\log n) \tag{88}$$

El cuarto método genera árboles aleatorios de diámetro $O(\log n)$ con $O(\log n)$ ramas.

IV.5 Análisis del tiempo de ejecución en cada tipo de árbol aleatorio

La Tabla V muestra en resumen el diámetro esperado de cada árbol utilizando cada uno de los diferentes métodos.

Tabla V: Comparación árboles aleatorios generados por cada método

Tipo	Diámetro	Ramas
1	$O(\log n)$	$O(n)$
2	$O(n^{\frac{1}{2}})$	$O(n^{\frac{1}{2}})$
3	$O(\log \log n)$	$O(n^{\frac{1}{2}})$
4	$O(\log n)$	$O(\log n)$

IV.5.1 Tiempo de ejecución del algoritmo de Xu y Srimani (Xu y Srimani, 2005) en árboles Tipo 1

En el capítulo III se demuestra que para un árbol arbitrario de diámetro $O(\log n)$, el tiempo promedio de ejecución es $O(n \log^2 n)$. Por lo tanto un árbol generado aleatoriamente usando el método uno se estabiliza en $O(n \log^2 n)$ pasos con alta probabilidad.

IV.5.2 Tiempo de ejecución del algoritmo de Xu y Srimani (Xu y Srimani, 2005) en árboles Tipo 2

Recuerde que el diámetro es $O(n^{\frac{1}{2}})$ y el número de ramas es $O(n^{\frac{1}{2}})$, por lo tanto hay a lo más $O(n^{\frac{1}{2}})$ nodos por nivel. Para calcular el tiempo de ejecución del algoritmo en un árbol Tipo 2 se toma el diámetro del árbol $O(n^{\frac{1}{2}})$ y se modifica la Ecuación 10 a

$$T(n) = \sum_{i=1}^{\lceil n^{\frac{1}{2}} \rceil} T_i = \ln 2 \sum_{i=1}^{\lceil n^{\frac{1}{2}} \rceil} (z_i - y_i) \log y_i + y_i \tag{89}$$

Suponga un árbol arbitrario Tipo 2 donde se privilegían candidatos de R_1 y R_2. Como se privilegían ambos tipos de candidatos, la ecuación se acota por arriba a $n \log n + n$ ya que para estabilizar cualquier externivel i pueden existir hasta n candidatos. Por lo tanto, el número de pasos promedio que se necesitan para estabilizar al árbol con respecto a R_1 es

$$T(n) = \ln 2 \sum_{i=1}^{\lceil n^{\frac{1}{2}} \rceil} (z_i - y_i) \log y_i + y_i \leq \ln 2 \sum_{i=1}^{\lceil n^{\frac{1}{2}} \rceil} n \log n + n \tag{90}$$

$$T(n) \leq \ln 2 \sum_{i=1}^{\lceil n^{\frac{1}{2}} \rceil} n \log n + n \tag{91}$$

$$T(n) \leq \ln 2 \left[n \log n \sum_{i=1}^{\lceil n^{\frac{1}{2}} \rceil} 1 + n \sum_{i=1}^{\lceil n^{\frac{1}{2}} \rceil} 1 \right] \tag{92}$$

$$T(n) \leq \ln 2 \left[n^{\frac{1}{2}} (n \log n) + n^{\frac{1}{2}} (n) \right] \tag{93}$$

El número de pasos promedio necesarios para estabilizar un árbol Tipo 2 con respecto a R_1 (dado que se permiten privilegiar candidatos de R_1 y R_2) es $O(n^{\frac{3}{2}} \log n)$. De la

misma manera se calcula el número de pasos promedio necesarios para estabilizar al árbol Tipo 2 con respecto a R_2 suponiendo que el árbol es estable con respecto a R_1. El número de pasos que se necesitan para estabilizar al árbol con respecto a R_2 no es mayor que el número de pasos para estabilizarlo con respecto a R_1.

IV.5.3 Tiempo de ejecución del algoritmo de Xu y Srimani (Xu y Srimani, 2005) en árboles Tipo 3

En el capítulo III se demuestra que para un árbol de profunidad logarítmica doble de diámetro $O(\log \log n)$, el tiempo promedio de ejecución es $O(n \log^2 n)$. Por lo tanto un árbol generado aleatoriamente usando el método tres se estabiliza en $O(n \log^2 n)$ pasos con alta probabilidad.

IV.5.4 Tiempo de ejecución del algoritmo de Xu y Srimani (Xu y Srimani, 2005) en árboles Tipo 4

En el capítulo III se demuestra que para un árbol arbitrario de diámetro $O(\log n)$, el tiempo promedio de ejecución es $O(n \log^2 n)$. Por lo tanto un árbol generado aleatoriamente usando el método cuatro se estabiliza en $O(n \log^2 n)$ pasos con alta probabilidad.

La Tabla VI muestra los tiempos de ejecución del algoritmo de Xu y Srimani (Xu y Srimani, 2005) para cada tipo de árbol generado aleatoriamente

Tabla VI: Tiempo promedio de ejecución en árboles arbitrarios

Árbol	Diámetro	$T(n)$
Tipo 1	$O(\log n)$	$O(n \log^2 n)$
Tipo 2	$O(n^{\frac{1}{2}})$	$O(n^{\frac{3}{2}} \log n)$
Tipo 3	$O(\log \log n)$	$O(n \log^2 n)$
Tipo 4	$O(\log n)$	$O(n \log^2 n)$

Capítulo V

Ejecución del algoritmo en un árbol binario balanceado utilizando ejecuciones paralelas sin calendarizador

Cuando se utiliza el calendarizador justo la ejecución del algoritmo es estrictamente secuencial. Dado que en un sistema distribuido varios nodos pueden ejecutar operaciones en paralelo, se propone modificar el algoritmo de Xu y Srimani (Xu y Srimani, 2005) para que se ejecute en paralelo. Se proponen dos modificaciones, la primera es a la regla R_1 donde se agrega una variable binaria b que toma el valor de uno o cero de forma aleatoria y con la misma probabilidad, los nodos i candidatos para R_1 se privilegían si $b_i = 1$ y cada nodo candidato j, vecino de i, tienen su variable $b_j = 0$, rompiendo así la simetría entre nodos. La segunda modificación es para la regla R_2, donde el nodo i candidato de R_2 sólo se privilegía si su nodo vecino j con $x_j \geq x_i$ no es candidato. Estas modificaciones implican eliminar por completo el calendarizador del análisis.

V.1 Estabilización con respecto a R_1 en un árbol binario balanceado utilizando ejecuciones en paralelo

Suponga un árbol binario balanceado. Los nodos con externivel 0 no pueden ser candidatos de R_1, los nodos del externivel i sólo pueden ser estables si todos los nodos de cada externivel $j < i$ son estables. Los nodos del externivel i sólo tienen conflictos de simetría con los nodos del externivel $i + 1$. Los nodos con conflictos deben generar aleatoriamente un uno o cero en su variable binaria. La probabilidad de que un nodo del externivel i tome en su variable binaria $b = 1$ y que su vecino del externivel $i + 1$ tome en su variable binaria $b = 0$ es de $\frac{1}{4}$.

El proceso de estabilización de este algoritmo es el mismo que cuando se usa el calendarizador justo en el Capítulo III, los nodos con externivel i se estabilizan sólo si los nodos con externivel $i-1$ son estables. Si y_i es el número de nodos en algún externivel i, el número esperado de nodos privilegiados en ese externivel en la primera iteración es $\frac{y_i}{4}$, por lo tanto faltan $\frac{3y_i}{4}$ nodos por privilegiar. El número de nodos que no se han privilegiado en la k-ésima iteración es $(\frac{3}{4})^k \times y_i$. Eventualmente, en la iteración α quedará solamente un nodo por privilegiar. Para calcular cuál es el número de iteraciones α que se requieren en promedio para privilegiar todos los y_i nodos excepto uno, se resuelve la ecuación,

$$\left(\frac{3}{4}\right)^{\alpha}(y_i) = 1 \tag{94}$$

$$\log\left(\frac{3}{4}\right)^{\alpha} = \log\left(\frac{1}{y_i}\right) \tag{95}$$

$$\alpha = \frac{\log 1 - \log y_i}{\log 3 - \log 4} \tag{96}$$

$$\alpha = \frac{\log y_i}{\log 4 - \log 3} \tag{97}$$

$$\alpha = C_1 \log y_i \tag{98}$$

En la Ecuación 98, α es el número de iteraciones que se requieren en promedio para que quede un nodo sin privilegiar, adicionalmente se requieren en promedio 4 pasos para privilegiar el último nodo. Conociendo el número de pasos que se requieren para privilegiar todos los nodos de algún externivel i, se puede calcular el número de pasos $T(n)$ que se requieren para estabilizar todo el árbol completo resolviendo la siguiente ecuación

$$T(n) = \sum_{i=1}^{\lceil \log n \rceil} C_1 \log y_i + C_2 \tag{99}$$

En la Ecuación 99, $C_1 \log y_i$ son los pasos necesarios para privilegiar todos los nodos candidatos y_i excepto el último y C_2 son los pasos necesarios para privilegiar el último. Resolviendo la siguiente ecuación se obtiene el número esperado de pasos para estabilizar al árbol.

$$T(n) = \sum_{i=1}^{\lceil \log n \rceil} C_1 \log y_i + \sum_{i=1}^{\lceil \log n \rceil} C_2 \tag{100}$$

$$T(n) = C_1 \sum_{i=1}^{\lceil \log n \rceil} \log \frac{2^i}{4} + C_2 \sum_{i=1}^{\lceil \log n \rceil} 1 \tag{101}$$

$$T(n) = C_1 \sum_{i=1}^{\lceil \log n \rceil} [\log 2^i - \log 4] + C_2 \sum_{i=1}^{\lceil \log n \rceil} 1 \tag{102}$$

$$T(n) = C_1 \sum_{i=1}^{\lceil \log n \rceil} [i - 2] + C_2 \log n \tag{103}$$

$$T(n) = C_1 \left(\frac{\log^2 n + \log n}{2} - 2 \log n \right) + C_2 \log n \tag{104}$$

$$T(n) = O(\log^2 n) \tag{105}$$

Teorema 8. *Sea T un árbol binario balanceado de n hojas y diámetro $O(\log n)$. El número de pasos promedio que tarda T para ser estable con respecto a R_1 es $O(\log^2 n)$ utilizando ejecuciones en paralelo.*

Comparando el Teorema 8 con el Teorema 5, se puede ver cómo el diámetro del árbol sigue siendo un factor que determina el tiempo de ejecución, y además, como las ejecuciones en paralelo ayudan a estabilizar más rápido cada externivel del árbol.

V.2 Estabilización con respecto a R_2 en un arbol binario balanceado utilizando ejecuciones en paralelo

Suponga un árbol binario balanceado estable con respecto a R_1 y donde todos los nodos con internivel mayor de cero son candidatos a R_2. Para este caso se propone que un nodo i candidato de R_2 sólo se puede privilegiar si su único vecino j con $x_j \geq x_i$ no es candidato. El máximo global con internivel 0 no puede ser candidato de R_2, por lo tanto los nodos del internivel 1 se pueden privilegiar ya que su único vecino mayor no es candidato, mientras que los candidatos de R_2 en cualquier internivel mayor no se pueden privilegiar. Todos los candidatos de R_2 en el internivel 1 se privilegían simultaneamente en un solo paso sin ocasionar conflictos ya que no son vecinos entre sí; una vez que esto suceda, los nodos del internivel 0 y 1 ya no son candidatos, por lo tanto los nodos del internivel 2 tambíen se privilegían simultaneamente en un solo paso. Esto sucede a través de todos los $O(\log n)$ interniveles del árbol.

Teorema 9. *Sea T un árbol binario balanceado de n hojas de diámetro $O(\log n)$ y estable con respecto a R_1. El número de pasos que tarda T para ser estable con respecto a R_2 es $O(\log n)$ utilizando ejecuciones en paralelo.*

V.3 Combinación de R_1 y R_2 en un árbol binario balanceado con ejecuciones en paralelo

Sea y_i el número de candidatos para R_1 y sea y'_i el número de candidatos para R_2 en el externivel i. Un árbol sólo puede ser estable si es estable con respecto a R_1 y R_2, y no puede ser estable con respecto a R_2 si primero no es estable con respecto a R_1. Mientras se estabiliza el externivel i, la cantidad de candidatos de R_2 que se privilegían junto con los candidatos de R_1 no afecta el número de pasos que se necesitan para privilegiar todos los y_i candidatos de R_1, y una vez que se terminan de privilegiar todos los y_i candidatos de R_1 no importa cuantos candidatos de R_2 faltan por privilegiarse, el proceso de estabilización puede seguir con el siguiente externivel.

Una vez que todo el árbol es estable con respecto a R_1 el proceso de estabilización con respecto a R_2 puede seguir sin tomar en cuenta el número de candidatos de R_2 que se privilegiaron antes, es por ésto que el número de pasos necesarios para estabilizar al árbol es igual a la suma del número de pasos necesarios para estabilizarlo con respecto a R_1 y R_2, es decir $\log^2 n + \log n = O(\log^2 n)$. En la sección 1 de este capítulo se calculan cuantos pasos se requieren para estabilizar un árbol con respecto a R_1 y en la sección 2 se calculan cuantos pasos se requiren para estabilizar un árbol con respecto a R_2.

Teorema 10. *Sea T un árbol binario balanceado de n hojas. El número de pasos promedio que tarda T para ser estable es $O(\log^2 n)$ utilizando ejecuciones en paralelo.*

V.4 Ejecución del algoritmo en un árbol de profundidad logarítmica doble utilizando ejecuciones paralelas sin calendarizador

Suponga un árbol de profundidad logarítmica doble. Los nodos con externivel 0 no pueden ser candidatos de R_1, los nodos del externivel i sólo pueden ser estables si todos los nodos de cada externivel $j < i$ son estables. Los nodos del externivel i sólo tienen conflictos de simetría con los nodos del externivel $i + 1$. Los nodos con conflictos deben generar aleatoriamente un uno o cero en su variable binaria. La probabilidad de que un nodo del externivel i tome en su variable binaria $b = 1$ y que su vecino del externivel $i + 1$ tome en su variable binaria $b = 0$ es de $\frac{1}{4}$.

El proceso de estabilización de este algoritmo es el mismo que cuando se usa el calendarizador justo en el Capítulo III, los nodos con externivel i se estabilizan sólo si los nodos con externivel $i - 1$ son estables. Si y_i es el número de nodos en algún externivel i, el número esperado de nodos privilegiados en ese externivel en la primera iteración es $\frac{y_i}{4}$, por lo tanto faltan $\frac{3y_i}{4}$ nodos por privilegiar. El número de nodos que no se han privilegiado en la k-ésima iteración es $(\frac{3}{4})^k \times y_i$. Eventualmente, en la iteración α quedará solamente un nodo por privilegiar. Para calcular cuál es el número de iteraciones α que se requieren en promedio para privilegiar todos los y_i nodos excepto uno, se resuelve la ecuación,

$$\left(\frac{3}{4}\right)^{\alpha} (y_i) = 1 \tag{106}$$

$$\alpha = \frac{\log y_i}{\log 4 - \log 3} \tag{107}$$

$$\alpha = C_1 \log y_i \tag{108}$$

Note cómo se llega al mismo resultando en la Ecuación 98 y (108). Se necesitan $O(\log y_i)$ pasos para privilegiar todos los candidatos con externivel 1. De la misma manera, después de que pasen este número de pasos, los nodos con externivel 1 ya no pueden ser candidatos de R_1, por esta razón los nodos con externivel 2 sólo tienen conflictos con el único vecino que tienen con externivel 3. Esto sucede a través de todos los $\log\log n$ externiveles del árbol, de tal forma que una vez que los nodos con externivel $i-1$ sean estables, los nodos con externivel i solo tienen conflictos con los nodos con externivel $i+1$.

Así como se hizo para el árbol binario balanceado se resuelve la siguiente ecuación para calcular el número de pasos promedio que se necesitan para estabilizar el árbol de profundidad logarítmica doble.

$$T(n) = \sum_{i=1}^{\lceil \log\log n \rceil} [C_1 \log y_i + C_2] \leq C_1 \log y_1 \sum_{i=1}^{\lceil \log\log n \rceil} 1 + C_2 \sum_{i=1}^{\lceil \log\log n \rceil} 1 \tag{109}$$

$$T(n) = O(\log n \log\log n) \tag{110}$$

Teorema 11. *Sea T un árbol de profundidad logarítmica doble con n hojas y diámetro $O(\log\log n)$. El número de pasos promedio que tarda T para ser estable es $O(\log n \log\log n)$ utilizando ejecuciones en paralelo.*

Note como el resultado del Teorema 11 es mejor que el resultado del Teorema 1 gracias a las ejecuciones paralelas.

Este análisis se puede generalizar a árboles arbitrarios de diámetro k. Dado que existen k externiveles en algún árbol, y que cada externivel necesita en promedio $O(\log n)$ pasos para ser estable, entonces el número de pasos total se puede acotar a $O(k \log n)$ y así obtener el siguiente teorema.

Teorema 12. *Sea T un árbol arbitrario de diámetro k. El número de pasos promedio que tarda T para ser estable es $O(k \log n)$ utilizando ejecuciones en paralelo.*

Capítulo VI

Conclusiones

Con este trabajo de tesis se llegan a las siguientes conclusiones:

1. Utilizando el calendarizador justo propuesto, el tiempo de ejecución promedio del algoritmo auto-estabilizante para elección de líder de Xu y Srimani (2005) es mucho menor a $O(n^4)$.

2. Desde un punto de vista teórico, la implementación del calendarizdor justo es factible (hay que recordar que el calendarizador adversario no fue diseñado para su implementación).

3. Utilizando el calendarizador justo en cualquier tipo de árbol, el número de pasos promedio que se requiere para estabilizar algún externivel o internivel con respecto a cualquier regla está dado por la Ecuación 8.

4. Dos factores principales definen el tiempo de ejecución del algoritmo. Por un lado está el número de pasos necesarios para estabilizar al árbol con respecto a R_1, la regla R_1 es quien define qué nodo será el líder. Por otro lado, el diámetro de un árbol influye en el tiempo de ejecución del algoritmo, esto se ve en la Ecuación 10.

5. Aún cuando no se conoce la topología precisa del árbol, es posible obtener un tiempo de ejecución bastante rápido como lo muestra el Capítulo IV con los árboles arbitrarios aplicando la Ecuación 10.

6. Las modificaciones al algoritmo que se proponen en el Capítulo V hacen posible eliminar completamente el calendarizador, ya que las reglas modificadas son capaces de permitir ejecuciones en paralelo sin causar conflictos y obtener así una aceleración considerable en el tiempo de ejecución.

7. En este trabajo de tesis se presentan varios modelos para crear árboles aleatorios, los cuales se pueden usar en la práctica para formar redes de computación. En estos casos el algoritmo estabiliza al árbol con un tiempo de ejecución bastante rápido.

8. Aunque los procesos que se proponen en este trabajo de tesis son probabilísticos, gracias a los análisis hechos en el Capítulo III se puede ver que el tiempo de ejecución promedio ocurre con bastante frecuencia cuando se utiliza el calendarizador justo.

Capítulo VII

Trabajo a futuro

Algunos puntos que se prestan a investigación adicional son:

1. La regla R_1 es la que escoge al líder, de tal manera que, después de tener un árbol estable con respecto a R_1 ya se puede conocer el líder. La tarea sería entonces evaluar las ventajas de las otras dos reglas del algoritmo y determinar si son necesarias para obtener estabilización.

2. Cada nodo sólo puede ver el estado de sus vecinos, pero se puede utilizar la información de los nodos de un nivel bajo para que los nodos con nivel alto tomen decisiones. Por ejemplo, si existe un candidato de R_2 en el internivel 1 y otro candidato de R_2 en el internivel 10, es posible que el candidato del internivel más alto tomo decisiones en base a la información del internivel bajo, de tal manera que los nodos del internivel alto no hagan operaciones adicionales y gasten el tiempo del calendarizador. Es posible utilizar este tipo de mensajes y calcular si el desempeño del algoritmo mejora o no.

3. Cuando se permiten ejecuciones en paralelo, el tiempo de ejecución óptimo es igual al externivel máximo, pero en el caso estudiado en este trabajo de tesis no es el caso. Una investigación a futuro puede ser cómo modificar el algoritmo de tal manera que se logre el tiempo de ejecución óptimo cuando se permiten ejecuciones en paralelo.

Bibliografía

Antonoiu, G. y Srimani, P. 1996. "A self-stabilizing leader election algorithm for tree graphs". Journal of Parallel and Distributed Computing. 34(2):227–232 p.

Antonoiu, G. y Srimani, P. 1997. "A self-stabilizing distributed algorithm to find the center of a tree graph". Parallel Algorithms and Applications. 10:237–248 p.

Burns, J. 1989. "Uniform self-stabilizing rings". ACM Transactions on Programming Languages and Systems. 11(2):330-344 p.

Dijkstra, E. W. 1974. "Self-stabilizing systems in spite of distributed control". Communications of the ACM. 17(11):643–644 p.

Dijkstra, E. W. 1986. "A belated proof of self-stabilization". Distributed Computing. 1(1):5–6 p.

Dolev, S., Israelí, A., y Moran, S. 1995. "Uniform self-stabilizing leader election part 2: General graph protocol". Technical report, Technion.

Fich, F. y Johnen, C. 2001. "A space optimal, deterministic, selt-stabilizing, leader election algorithm for unidirectional rings". En: "Proc. of the 15th International Symposium on Distributed Computing", Lisbom, Portugal. 224–239 p.

Ghosh, S. 2001. "Cooperating mobile agents and stabilization". En: "WSS '01: Proceedings of the 5th International Workshop on Self-Stabilizing Systems", Londres, Inglaterra. 1–18 p.

Ghosh, S. y Gupta, A. 1996. "An exercize in fault-containment: Self-stabilizing leader election". Information Processing Letters. 59(5):281–288 p.

Goddard, W., Hedetniemi, S., Jacobs, D., y Srimani, P. 2005. "Self-stabilizing algorithms for orderings and colorings". Internacional Journal of Foundations of Computer Science. 16(1):19-36 p.

Gradinariu, M. y Tixeuil, S. 2000. "Self-stabilizing vertex coloration and arbitrary graphs". En: "OPODIS". 55-70 p.

Huang, S. 1993. “Leader election in uniform rings”. ACM Transactions on Programming Languages and Systems. 15(3):563–573 p.

Kessels, J. 1988. “An exercise in proving self-stabilization with a variant function”. Information Processing Letters. 29(1):39–42 p.

Motwani, R. y Raghavan, P. 1995. “Randomized algorithms”. Cambridge University Press. Primera edición. 57, 58 pp.

Schneider, M. 1993. “Self stabilization”. ACM Computing Surveys. 25(1):45–67 p.

Xu, Z. y Srimani, P. 2005. “Self-stabilizing anonymous leader election in a tree”. En: “Proceedings of the 19th IEEE IPDPS”, Washington, EEUU. 207–214 p.

Vita

Juan Paulo Alvarado Magaña obtuvo el título de Ingeniero en Computación en la Universidad Autónoma de Baja California campus Tijuana al obtener el certificado de Desempeño Académico Satisfactorio del Centro Nacional de Evaluación en el año 2001. Posteriormente se desempeñó en el campo industrial desarrollando aplicaciones para empresas como Hewlett Packard y Panasonic. Del año 2004 al 2006 cursó la Maestría en Ciencias de la Computación en el Centro de Investigación Científica y de Educación Superior de Ensenada, Baja California. Sus áreas de interés incluyen: Inteligencia Computacional, Cómputo Paralelo y Distribuido y Generación de Imágenes.