20

Universidad Nacional Autónoma de México

Facultad de Ingeniería

"El algoritmo de hill climbing para empujar información dependiente en sistemas tipo Clientes – Proveedores – Servidores"

T E S I S

que para obtener el título de:

Ingeniero en Telecomunicaciones

Módulo: Procesamiento Digital de Señales

P R E S E N T A:

Francisco Javier Ovalle Martínez

Director de Tesis: Dr. Ivan Stojmenovic

2 · 9606

Ciudad Universitaria 2001

Universidad Nacional Autónoma de México
UNAM
Biblioteca Central
Dirección General de Bibliotecas de la UNAM

UNAM – Dirección General de Bibliotecas
Tesis Digitales
Restricciones de uso

DERECHOS RESERVADOS ©
PROHIBIDA SU REPRODUCCIÓN TOTAL O PARCIAL

Todo el material contenido en esta tesis esta protegido por la Ley Federal del Derecho de Autor (LFDA) de los Estados Unidos Mexicanos (México).

El uso de imágenes, fragmentos de videos, y demás material que sea objeto de protección de los derechos de autor, será exclusivamente para fines educativos e informativos y deberá citar la fuente donde la obtuvo mencionando el autor o autores. Cualquier uso distinto como el lucro, reproducción, edición o modificación, será perseguido y sancionado por el respectivo titular de los Derechos de Autor.

Dedicatorias

A la Universidad Nacional Autónoma de México.

Con la promesa incondicional de ayudarla y defenderla para que siga siendo la principal forjadora de sueños, esperanzas y logros de México.

A mi madre, mi abuela, y mis hermanos.

Mami y Chata les agradezco mucho en que a pesar de las condiciones que encontramos, siempre me apoyaron y supieron llevarme a la persona que soy ahora, gracias con todo el corazón.

Irma y Alfonso, nunca se detengan para alcanzar sus sueños y recuerden que siempre estaremos juntos para ayudarnos.

A la generación 97 - 1 de Telecomunicaciones

Bertín, Rafael, Víctor C., Óliver, Perla, Iván, Óscar, Salvador, Alejandro, Jaime, Osvaldo, Frank, Daniel, Rodolfo, Roberto, Alfonso, Carlos, Víctor R., Omar, Karla, Juan Carlos, Silvia, Juan, José y Lida.

A mis amigos

Iris, Sergio, Drew, Ramón, Trini, Gilberto, Mariano, Hida, Carlos R., Silvia T., Carolina, Alfonso, Silvia F., Óscar, Frank, Denisse, Adriana, Víctor C., Óliver, Roberto, Marisol, Ricardo, Lalo M., Luis, Lalo G. y, Jorge.

A las familias
Campos Carcaño y Chong Lee

Gracias por recibirme y tratarme como a uno más de ustedes

Iris

Gracias por brindarme tu amistad, tu alegría y tu apoyo en estos 5 años de tomar clases, de estudiar, de hacer tareas y trabajos, de desvelos, de jugar basket, de ir a fiestas, de ir al cine, de platicar, de oír música, de ir al gimnasio, de verte bailar jazz y danza, de tantas locuras y humor que me alegraron muchísimo la vida y lograron animarme cuando lo necesité, eres lo mejor que me llevo de esta Universidad pues hemos logrado una "amistad" a tal grado que casi nadie llega a comprender, te quiero muchísimo

Kengtin

Te agradezco estos años de amistad, de ayuda incondicional y de recuerdos. gracias por ser tan buen amigo. has sido parte fundamental para ser la persona que soy. gracias por todo hermano

A los profesores

Enrique Arenas Sánchez
Leonardo Bañuelos Saucedo
Francisco Barrera García
Érik Castañeda De Isla Puga
Margarita Ramírez Galindo
Jorge Solar González
Mario Alfredo Ibarra Pereyra
Gloria Matu Hernández
Juan José Origel Coutiño
Carlos Rivera Rivera
Manuel Sánchez Esquivel
Rodolfo Neri Vela
Benjamín Valera Orozco
Jesús Reyes García
Ricardo Mandujano Wild
Bohumil Psenicka
Lorenzo Octavio Miranda Cordero

¡Gracias por su excelencia académica

A los que se han ido
Papá, abuelo Alfonso y
bisabuela Josefina

Nunca los olvidaré y tendré siempre como ejemplo a seguir las grandes cualidades que tuvieron en vida y que hicieron de ustedes personas admirables y queridas.

A. Ivan Stojmenovic

¡Gracias por su paciencia y su asesoría, sin usted no hubiera sido posible el desarrollo de este trabajo

Resumen

En redes inalámbricas y satelitales y en sistemas avanzados de información de tráfico en donde el ancho de banda de enlace de subida es muy limitado, un servidor realiza un broadcast de archivos de información en forma cíclica. Los archivos de información son distribuidos por diferentes proveedores y son accesados por muchos clientes. Los proveedores son independientes y por lo tanto algunos archivos pueden compartir información. Los clientes que accesan estos archivos pueden tener diferentes patrones de acceso. Algunos clientes pueden desear accesar mas de un archivo en cualquier orden, algunos pueden accesar un archivo de varios existentes, y algunos clientes pueden accesar un segundo archivo solo después de accesar otro archivo. El objetivo del servidor es de ordenar los archivos de tal forma que se minimice el tiempo de acceso de los clientes dado algún conocimiento a priori de sus patrones de acceso. En este trabajo se propone una forma de que el servidor ordene los archivos mediante el algoritmo de Hill Climbing y de esta forma mejorar los métodos ya existentes para ordenar estos archivos.

Índice

Título	Página
Objetivos	1
Capítulo I	2
1.1 Introducción	3
1.2 Planteamiento	4
1.3 Resultados existentes	6
1.4 El algoritmo de Hill Climbing	8
1.5 El algoritmo de Hill Climbing como solución alterna	9
Capítulo II	14
2.1 Transmisión tipo broadcast	15
2.2 Empujando información dependiente en sistemas Clientes – Proveedores – Servidores	17
2.2.1 Introducción	17
2.2.2 Definiciones	19
2.2.3 Desarrollo	21
2.3 El Algoritmo de Hill Climbing	24
2.3.1 Algunos operadores para permutaciones	25
2.3.2 Hill Climbing	25
Capítulo III	28
3.1 La evolución de las permutaciones por medio del operador intercambio y su relación con las matrices de distancias dij y distancias directas ddij	29
Capítulo IV	36
4.1 Análisis y Simulación	37
4.1.1 Generación de las variables usadas en la simulación	37
- Generación de las permutaciones π	37
- Generación de las matrices de probabilidades a_{ii}, b_{ij} y c_{ij}	38
4.1.2 Implementación del algoritmo de Hill Climbing	41

4.1.3 Primer simulación: Comparación de los resultados finales del algoritmo de hill climbing con los del algoritmo aleatorio.. 43
4.1.4 Segunda simulación: Comparación del tiempo de acceso entregado por el algoritmo de hill climbing y el algoritmo aleatorio con tiempos semejantes de procesamiento..... 46
4.1.5 Tercera simulación: Comparación del tiempo de acceso y el tiempo de procesamiento entregado por el algoritmo de hill climbing y el algoritmo aleatorio con un número dado de permutaciones a evaluar..... 51

Capítulo V 56
5.1 Conclusiones y futuros trabajos 57
5.2 Referencias 60

Apéndice A: Resultados y Gráficas adicionales..... 61

Apéndice B: Código 66

Objetivos

- Aplicación del algoritmo de Hill Climbing en sistemas asimétricos de telecomunicaciones tipo Clientes - Proveedores - Servidores.

- Mejoramiento del tiempo de acceso de los usuarios en los sistemas asimétricos de telecomunicaciones tipo Clientes - Proveedores - Servidores.

- Comparación del tiempo de procesamiento del algoritmo de Hill Climbing y el tiempo de procesamiento obtenido mediante la generación de un número aleatorio de cartas de programación.

Capítulo I

Este capítulo tiene como objetivo el mostrar la idea y los resultados generales del trabajo desarrollado en la presente tésis. De esta manera la persona que lo lea, podrá consultar de una forma rápida el problema planteado y si desea profundizar más en el tema, podrá leer los capítulos consecuentes.

1.1 Introducción

En redes satelitales e inalámbricas y en sistemas avanzados de información de tráfico en donde el ancho de banda del canal de subida es muy limitado,un servidor transmite los archivos de información de manera cíclica. Los archivos de información son suministrados por diferentes proveedores y son accesados por muchos clientes. Los proveedores son independientes y por lo tanto los archivos pueden compartir información. Los clientes que accesan a estos archivos pueden tener diferentes patrones de acceso. Algunos clientes pueden desear accesar más de un archivo en un momento dado en cualquier orden, algunos clientes pueden accesar un archivo de varios disponibles, y algunos clientes pueden desear accesar un segundo archivo sólo después de accesar otro. El objetivo del servidor es de ordenar los archivos de tal forma que se minimize el tiempo de acceso de los clientes teniendo algún conocimiento a-priori de sus patrones de acceso.

Muchas nuevas aplicaciones en redes tienen la propiedad de que el canal de comunicación entre el servidor y el cliente es asimétrico. Esto significa que existe mucho mayor ancho de banda desde el servidor hacia el cliente que en la dirección contraria. Por lo tanto la información es diseminada (empujada) desde un servidor a muchos clientes en contraste con aplicaciones tradicionales en donde la información se jala desde los servidores hacia las aplicaciones de los clientes conforme se realizan las peticiones. Una solución propuesta para canales de comunicaciones asimétricos es la de usar sistemas de empuje de la información. La idea central es que el servidor explote su ventaja en el ancho de banda transmitiendo la información a múltiples clientes. Aquí, el servidor continuamente y repetidamente transmite la información a una comunidad de clientes, de esta forma el canal de transmisión se convierte en un "disco" de donde los clientes pueden obtener la información conforme esta pasa.

En los teléfonos celulares y en las computadoras personales portátiles existe el problema de la potencia de las baterías que necesitan, desgraciadamente los fabricantes de dichas baterías se han quedado rezagados en desarrollar tecnologías que amplien su tiempo de uso real, es por esto que si se llegan a desarrollar técnicas que minimizen el tiempo de acceso de los usuarios también se estará minimizando la potencia consumida de los portátiles, consiguiendo de esta forma la eficiencia máxima de los aparatos portátiles inalámbricos.

En la actualidad los sistemas inalámbricos de telecomunicaciones tienen un papel fundamental en el desarrollo de las actividades de casi todos los seres humanos, la tendencia indica que todos estarán comunicados por medio de teléfonos celulares o computadoras tipo palms, interconectándose por supuesto a la red mundial, al internet. La importancia de esta intercomunicación será trascendente en casi todas las actividades humanas, destacando actividades económicas, políticas, sociales, en materia de educación, de salud, etc....

1.2 Planteamiento

El principal objetivo para el planteamiento del problema es de explorar diferentes formas en las que los servidores minimizen o mejoren el tiempo promedio de acceso de los clientes basándose en alguna información a-priori sin violar los contratos, por ejemplo sin cambiar las frecuencias de transmisión de archivos específicos, por frecuencias me refiero a que tan repetido se transmite un archivo y no a la frecuencia portadora de transmisión.

Como ya se mencionó existen tres patrones en la que los clientes accesan la información, de una manera *más general* solo se verá la interdependencia entre pares de archivos, cabe aclararse que en la realidad la dependecia entre archivos puede ser mayor y no debe restringirse a solo pares de archivos.

1.2.1 DEFINICIONES

Sea $F = \{F_0, F_1, ..., F_{n-1}\}$ una colección de n archivos de igual tamaño.

A continuación se definen las probabilidades para los diferentes problemas que se consideran:

- En el problema **AND**, para $0 \le i \le j \le n-1$, la probabilidad de accesar F_i y F_j es a_{ij}.

La probabilidad de accesar solo F_i es a_{ii}. Existen $\frac{n(n+1)}{2}$ diferentes probabilidades y $\sum_{j=0}^{n-1}\sum_{i=0}^{i\le j} a_{i,j} = 1$ o de forma más general $\Sigma_{0 \le i \le j \le n-1} a_{ij} = 1$.

- En el problema **OR**, para $0 \le i \le j \le n-1$, la probabilidad de accesar F_i o F_j es b_{ij}.

La probabilidad de accesar solo F_i es b_{ii}. Existen $\frac{n(n+1)}{2}$ diferentes probabilidades y $\sum_{j=0}^{n-1}\sum_{i=0}^{i\le j} b_{i,j} = 1$ o de forma más general $\Sigma_{0 \le i \le j \le n-1} b_{ij} = 1$.

- En el problema **IMPLY**, para $0 \le i \ne j \le n-1$, la probabilidad de accesar F_i y después F_j es c_{ij}. La probabilidad de accesar solo F_i es c_{ii}. Existen n^2 diferentes probabilidades y $\sum_{j=0}^{n-1}\sum_{i=0}^{n-1} c_{i,j} = 1$ o de forma más general $\Sigma_{0 \le i,j \le n-1} c_{ij} = 1$.

La programación de un ciclo infinito se define como la permutación $\pi = (\pi(0), \pi(1), ..., \pi(n-1))$ en los números $(0, 1, ..., n-1)$. Sea $\sigma = \pi^{-1}$, es decir, $\sigma(\pi(i)) = i$ para toda $0 \le i$

$\leq n-1$. Se dice que los archivos F_i y F_j están a una distancia d_{ij} si:

$$d_{ij} = \min\{ ((\sigma(i) - \sigma(j)) \bmod n), ((\sigma(j) - \sigma(i)) \bmod n)\}$$

para $0 \leq i < j \leq n-1$.

Se dice que los archivos Fi y Fj están a una distancia directa si:

$$dd_{ij} = ((\sigma(j) - \sigma(i)) \bmod n)$$

para $0 \leq i, j \leq n-1$.

Para una permutación π, $A_{ij}(\pi)$ sea el tiempo esperado de espera para un cliente que desea accesar los archivos F_i y F_j, $B_{ij}(\pi)$ sea el tiempo esperado de espera para un cliente que desea accesar ya sea el archivo F_i o el archivo F_j, y sea $C_{ij}(\pi)$ el tiempo esperado de espera para un cliente que desea accesar primero el archivo F_i y solo después el F_j. Sea

$$A(\pi) = \Sigma_{0 \leq i \leq j \leq n-1}[(a_{ij})(A_{ij}(\pi))]$$

$$B(\pi) = \Sigma_{0 \leq i \leq j \leq n-1}[(b_{ij})(B_{ij}(\pi))]$$

$$C(\pi) = \Sigma_{0 \leq i \neq j \leq n-1}[(c_{ij})(C_{ij}(\pi))]$$

Esto es, $A(\pi)$ es el tiempo esperado para todos los clientes que desean accesar uno o dos archivos, $B(\pi)$ es el tiempo esperado para todos los clientes que desean accesar un archivo o uno de dos archivos, y $C(\pi)$ es el tiempo esperado para todos los clientes que desean accesar un archivo o dos archivos en cierto orden.

Después de cierto análisis matemático se obtienen las siguientes expresiones, las cuales se muestran su deducción en el capítulo 2:

$$A(\pi) = \frac{n}{2} + 1 + \sum_{j=0}^{n-1} \sum_{i=0}^{i \leq j} a_{ij} \left(d_{ij} - \frac{d_{ij}^2}{n}\right) \quad \text{.....(1)}$$

$$B(\pi) = \frac{n}{2} + 1 - \sum_{j=0}^{n-1} \sum_{i=0}^{i \leq j} b_{ij} \left(d_{ij} - \frac{d_{ij}^2}{n}\right) \quad \text{.....(2)}$$

$$C(\pi) = \frac{n}{2} + 1 + \sum_{j=0}^{n-1} \sum_{i=0}^{n-1} c_{ij} (dd_{ij}) \quad \text{.....(3)}$$

En sí para el problema AND se busca una permutación π que minimize $A(\pi)$, en el problema OR se busca una permutación π que minimize $B(\pi)$, y en el problema IMPLY se busca una permutación π que minimize $C(\pi)$. Si se logra minimizar estos tres casos se logrará minimizar el tiempo de acceso de los clientes.

1.3 Resultados existentes

Se han desarrollado dos métodos para encontrar las permutaciones que hagan mínimas las expresiones (1), (2) y (3), el primero se llama el algoritmo aleatorio y el algoritmo esquemático determinístico. A continuación se describen ambos.

Algoritmo Aleatorio.

Entradas:

n.- El número de archivos que se van a transmitir.
m.- El número de permutaciones aleatorias a checar.
X.- El problema a resolver.

x_{ij}.- Las probabilidades de las dependencias de los archivos.

Salidas:

Una permutación π que define el orden de los archivos.

Método:

Contruir un conjunto M de m permutaciones aleatorias.
Cada permutación con probabilidad $\frac{1}{n!}$
Para cada permutación t que pertenezca a M calcular X(t)
Escoger la permutación π para el cual $X(\pi)$ es mínima sobre todas las permutaciones en M.

El resultado de este algoritmo mejora conforme m aumenta, esto es claro ya que entre más permutaciones se tengan para evaluar las ecuaciones (1), (2) y (3) nos podremos acercar más al valor mínimo requerido, desgraciadamente mientras más grande sea tanto el tamaño y el número de las permutaciones a evaluar, el tiempo de procesamiento crece casi de forma cuadrática, esto se mostrará mas adelante, por lo cual este método es poco eficiente en cuanto al tiempo de procesamiento computacional.

Algoritmo Esquemático Determinístico.

Entradas:

n.- El número de archivos que se van a transmitir.
X.- El problema a resolver.
x_{ij}.- Las probabilidades de las dependencias de los archivos.
M.- Un conjunto de permutaciones dadas.

Salidas:

Una permutación π perteneciente a M que define el orden de los archivos.

Método:

Para cada permutación t que pertenezca a M calcular X(t)

Escoger la permutación π para el cual $X(\pi)$ es mínima sobre todas las permutaciones en M.

Este algoritmo difiere del anterior en que aquí se dan específicamente las permutaciones a evaluar, pero en esencia los dos algoritmos son iguales, evaluar las permutaciones dadas con las ecuaciones (1), (2) y (3) y elegir la que otorgue el valor mínimo, en este caso se garantiza obtener el valor mínimo si se dan como entrada las n! permutaciones posibles, desgraciadamente también es poco eficiente en cuanto al tiempo de procesamiento computacional y en sí no cambia mucho con respecto al algoritmo aleatorio, es por esto que este algoritmo no se analizará más en este trabajo pues creo que será suficiente con el algoritmo aleatorio para obtener resultados semejantes con ambos.

Como se observa, los algoritmos anteriores implican un gran número de operaciones por lo cual son muy ineficientes para programarlos ya que el tiempo de procesamiento de ambos es muy grande debido a la generación de las permutaciones, y al cálculo de las fórmulas (1), (2) y (3) para cada permutación.

1.4 El algoritmo de Hill Climbing

Debido a los problemas presentados por los algoritmos anteriores se presenta una solución alterna, el algoritmo de hill climbing, este algoritmo es una técnica heurística. Heurístico significa métodos, criterios o principios para decidir entre varios caminos alternos de decisiones o acciones usadas para determinar soluciones que toman tiempo infinito o tiempo muy extenso en procesamiento computacional. Estos métodos no siempre encuentran la mejor solución, pero garantizan encontrar una muy buena en un tiempo razonable.

El algoritmo de hill climbing consiste en evaluar la solución actual y "moverse" a una mejor solución más cercana al objetivo. Para determinar si otra solución es mejor se necesita algún tipo de evaluación o función. La solución se alcanza cuando la función a evaluar es mínima. Esta función se conoce como función de propiedad. En este caso tenemos tres funciones de propiedad, la funciones (1), (2) y (3).

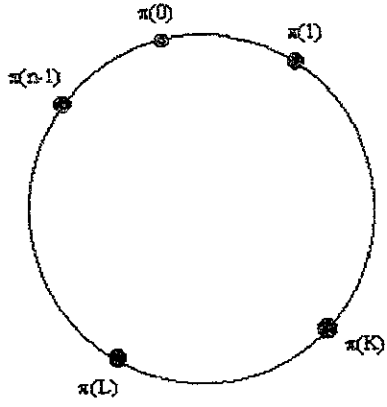
Primero se define una solución inicial, digamos So y calculamos la función de propiedad de So. Después se define uno, algunos o todos los vecinos de So. Los vecinos de una permutación se obtienen aplicando un operador computacional evolutivo a la permutación candidata, en este caso So. En este momento no se explicarán los operadores computacionales evolutivos, estos se tratarán en el capítulo 3. A esos vecinos obtenidos se les aplica la función de propiedad y se elige la de menor valor, este proceso se repite hasta que no se encuentra mejoría en las soluciones, cabe hacer notar que el algoritmo también puede aplicarse a problemas donde se quiere encontrar un máximo.

1.5 El algoritmo de Hill Climbing como solución alterna

En este punto explicaré a groso modo la forma en que apliqué el algoritmo de hill climbing para dar una solución alterna al problema planteado, las ecuaciones (1), (2) y (3) son las funciones de propiedad que usé para aplicar el algoritmo de hill climbing.

Una permutación π es un ciclo de archivos ordenados de cierta forma, esto es:

$\pi = \{\pi(0), \pi(1), \pi(2), \ldots, \pi(n-1)\}$



Como no podemos repetir los archivos que son más populares, lo único que podemos hacer es reordenar la permutación π para minimizar $A(\pi)$, $B(\pi)$ y $C(\pi)$, si observamos las ecuaciones (1), (2) y (3) se nota que a_{ij}, b_{ij} y c_{ij} son fijos, n también es constante por lo que lo único que se puede cambiar son las distancias de los archivos, esto es, reordenar los archivos de la permutación π.

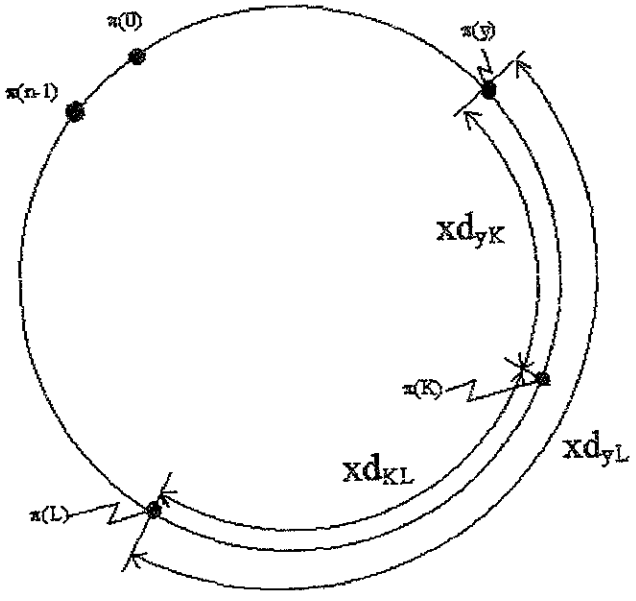
- Se calcula $X(\pi)$ (X puede ser A, B o C, según sea el caso).
- Se *intercambian* dos elementos de π y se calcula $X(\pi')$, π' es la nueva permutación.
- Se comparan $X(\pi')$ con $X(\pi)$ y se elige la mejor.
- Estos pasos se repiten hasta que no se encuentra mejora alguna.

Para calcular $X(\pi)$ se necesitan x_{ij} y xd_{ij}, donde x_{ij} es la matriz de probabilidades dependientes entre los pares de archivos i y j, y xd_{ij} es la matriz de distancias o de distancias directas entre los archivos i y j.

Para minimizar los cálculos de permutación a permutación al momento de realizar el intercambio de archivos se realizó lo siguiente:

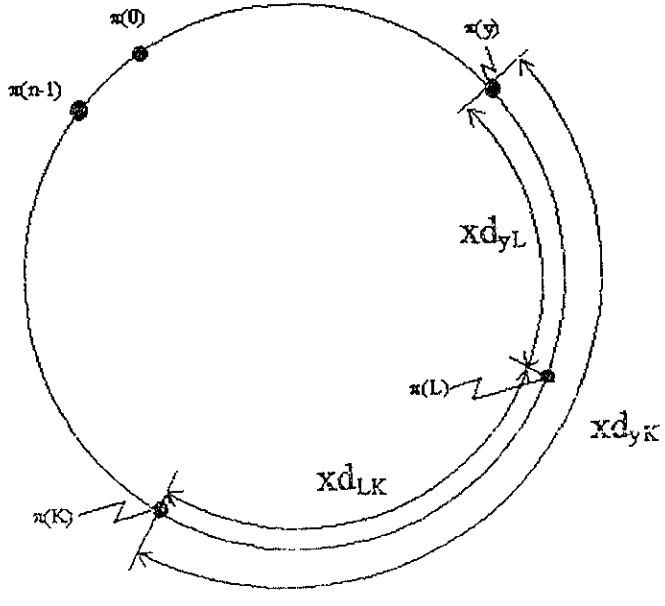
Figura 1.

π:



Si se intercambian los archivos K y L se obtiene:

Figura 2. π':



Observando las figuras 1 y 2 se nota que al momento de intercambiar los elementos L y K de la permutación π se llega a las siguientes relaciones:

xd_{yK} cambia a xd_{yL}
xd_{KL} cambia a xd_{LK}
xd_{yL} cambia a xd_{yK}

Viendo estas relaciones concluí que se debía intercambiar los renglones y las columnas de los archivos Ky L en la matriz xd_{ij}.

Ahora se mostrará la matriz xd_{ij}.

$$xdij = \begin{bmatrix} 0 & xd_{0,1} & \cdots & xd_{0,y} & \cdots & xd_{0,K} & \cdots & xd_{0,L} & \cdots & xd_{0,n-1} \\ xd_{1,0} & 0 & \cdots & xd_{1,y} & \cdots & xd_{1,K} & \cdots & xd_{1,L} & \cdots & xd_{1,n-1} \\ \vdots & \vdots & \cdots & \vdots & \cdots & \vdots & \cdots & \vdots & \cdots & \vdots \\ xd_{y,0} & xd_{y,1} & \cdots & 0 & \cdots & xd_{y,K} & \cdots & xd_{y,L} & \cdots & xd_{y,n-1} \\ \vdots & \vdots & \cdots & \vdots & \cdots & \vdots & \cdots & \vdots & \cdots & \vdots \\ xd_{K,0} & xd_{K,1} & \cdots & xd_{K,y} & \cdots & 0 & \cdots & xd_{K,L} & \cdots & xd_{K,n-1} \\ \vdots & \vdots & \cdots & \vdots & \cdots & \vdots & \cdots & \vdots & \cdots & \vdots \\ xd_{L,0} & xd_{L,1} & \cdots & xd_{L,y} & \cdots & xd_{L,K} & \cdots & 0 & \cdots & xd_{L,n-1} \\ \vdots & \vdots & \cdots & \vdots & \cdots & \vdots & \cdots & \vdots & \cdots & \vdots \\ xd_{n-1,0} & xd_{n-1,1} & \cdots & xd_{n-1,y} & \cdots & xd_{n-1,K} & \cdots & xd_{n-1,L} & \cdots & 0 \end{bmatrix}$$

Nótese que $xd_{ii} = 0$.

Por medio del intercambio de las columnas y los renglones de la matriz xdij no se tiene que calcular toda la nueva matriz para la permutación π'. Por ejemplo para el problema AND

$$A(\pi) = \frac{n}{2} + 1 + \sum_{j=0}^{n-1} \sum_{i=0}^{i \le j} a_{ij} (d_{ij} - \frac{d_{ij}^{\,2}}{n})$$

$$A(\pi') = A(\pi) - suma1 + suma2 + a_{LK} (d_{LK} - \frac{d_{LK}^{\,2}}{n}) - a_{LK} (d_{L'K'} - \frac{d_{L'K'}^{\,2}}{n})$$

$$suma1 = \sum_{j=0}^{K-1} a_{Kj} (d_{Kj} - \frac{d_{Kj}^{\,2}}{n}) + \sum_{j=0}^{L-1} a_{Lj} (d_{Lj} - \frac{d_{Lj}^{\,2}}{n}) + \sum_{i=K+1}^{n-1} a_{iK} (d_{iK} - \frac{d_{iK}^{\,2}}{n}) + \sum_{i=L+1}^{n-1} a_{iL} (d_{iL} - \frac{d_{iL}^{\,2}}{n})$$

suma2 se obtiene de la misma forma que suma1 solo que ahora se utilizará la nueva matriz de distancias d_{ij}' que se obtuvo intercambiando las columnas y los renglones K y L, debo hacer notar que para el cálculo de suma 2 se sigue utilizando la misma matriz de probabilidades a_{ij}.

De forma similar se desarrollan los casos OR e IMPLY, su explicación se encuentra en el capítulo 3.

A continuación de una forma muy general describo los pasos que seguí para aplicar el algoritmo de hill climbing:

- Generé un conjunto de t permutaciones aleatorias.
- Según el problema a tratar, generé las matrices de probabilidades entre archivos.
- En cada permutación apliqué el algoritmo de hill climbing para mejorarla, conté el número total de permutaciones evaluadas con las funciones de propiedad, escogí la mejor permutación P1 y guardé su tiempo de procesamiento.
- Generé un conjunto de permutaciones aleatorias igual al número total de permutaciones evaluadas en el punto anterior y apliqué el algoritmo aleatorio descrito anteriormente, escogiendo la mejor permutación P2, guardando también su tiempo de procesamiento.
- Comparé P1 y P2 con sus respectivos tiempos de procesamiento.

A continuación se muestra una tabla comparativa de los dos algoritmos.

Caso	*t*	*n*	**N° total de permutaciones evaluadas con ambos algoritmos**	***Tiempo de acceso de H.C.***	***Tiempo de acceso mediante el algoritmo aleatorio***	***Tiempo de procesamiento en segundos con H.C.***	***Tiempo de procesamiento en segundos con el algoritmo aleatorio***
AND	15	5	535	3.8002	3.8002	1.9933	21.4033
AND	30	10	2556	6.3331	6.3370	17.7000	528.1667
AND	45	15	5343	9.2579	9.4564	46.1567	2110.9333
AND	60	20	9314	11.8009	12.3500	86.0333	6304.4667
AND	75	25	14652	14.6831	15.6395	187.2400	16137
OR	15	5	522	2.9362	2.9362	1.8133	20.3
OR	30	10	2448	5.0986	5.1019	12.23	373.1
OR	45	15	5293	6.6188	6.6563	38.0267	1827.5
OR	60	20	8196	8.4513	8.5464	80.6301	5561.4
OR	75	25	11651	9.7356	9.9219	146.6499	11727.3
IMPLY	15	5	603	4.1181	4.1181	1.8100	12.5
IMPLY	30	10	2248	8.0140	8.1899	10.4167	178.9
IMPLY	45	15	5273	12.4061	12.6271	39.2033	1126.6
IMPLY	60	20	8661	16.6485	17.5803	69.4299	2722.5
IMPLY	75	25	11745	21.507	22.6458	118.8099	6377.1

De la tabla, el tiempo de procesamiento computacional disminuye de una forma muy significativa usando el algoritmo de hill climbing, además de que el tiempo de acceso obtenido es igual o menor al obtenido mediante el algoritmo aleatorio, este es solo un primer resultado, si se desea ver más resultados y un análisis más a fondo consulte el capítulo IV , si se quiere tener el código del programa para hacer sus propias pruebas consulte el apéndice B.

Capítulo II

Este capítulo tiene el fin de explicar con más detalle los conceptos del problema planteado así como la teoría que se necesitó para poder llegar a la solución del problema mediante el algoritmo de hill climbing.

2.1 Transmisión tipo broadcast.

Hemos sido testigos como en estos años han crecido de forma gigantesca las aplicaciones inalámbricas de la información en el mercado comercial gracias a la llegada de dispositivos inalámbricos, de redes inalámbricas de gran velocidad, y de software capaz de soportar estas tecnologías. En un futuro muy próximo, un gran número de usuarios móviles con dispositivos portátiles (como palmtops, laptops, PDAs, teléfonos WAP, etc.) serán capaces de accesar información desde cualquier lugar en cualquier momento. El tipo de información que puede ser accesado inalámbricamente puede ser ilimitado, e incluye noticias, precios de productos en existencia, programa de vuelos, información sobre tráfico vehicular, el clima, etc.

Existen dos métodos fundamentales para entregar la información en aplicaciones inalámbricas: acceso punto a punto y la transmisión masiva o broadcast. En el acceso punto a punto, un canal lógico se establece entre el cliente y el servidor. Las solicitudes se envían al servidor y los resultados se regresan al cliente en casi la misma manera que sucede en una red alámbrica. En el broadcast, la información se envía de manera simultánea a todos los usuarios que se encuentran en el área de recepción del broadcast. Es cuestión del cliente el seleccionar la información que desea. Comparado con el acceso punto a punto, el broadcast es un método más atractivo por las siguientes razones:

- Un solo broadcast de un elemento de información puede satisfacer todas las solicitudes por ese elemento simultánemente. Por lo tanto, el broadcast puede llegar a un número arbitrario de usuarios.

- Las comunicaciones móviles inalámbricas se caracterizan por ser comunicaciones asimétricas, es decir, el canal de comunicación de bajada es mucho mayor que el canal de comunicación de subida. La transmisión por broadcast puede tomar ventaja del ancho de banda del canal de comunicación de bajada cuando envía la información a los clientes.

- Un sistema de comunicación inalámbrica usa esencialmente un componente de broadcast para entregar la información. Por lo tanto, la transmisión de la información por medio de broadcast puede implementarse sin ningún costo extra.

La eficiencia del acceso y la conservación de la potencia son dos problemas fundamentales en cualquier sistema de información inalámbrico. La eficiencia en el acceso se refiere a que tan rápido se satisface una solicitud, mientras que la conservación de la potencia se refiere a como reducir el consumo de potencia del dispositivo móvil cuando accesa la información que desea. El segundo problema es muy importante por la potencia limitada de la batería de los dispositivos móviles, la cual varía de unas pocas horas hasta aproximadamente la mitad de un día bajo uso ininterrumpido. Además, solo se espera una mejoría del 20% al 30% en la capacidad de las baterías en los próximos años. En la literatura disponible, dos métricas básicas de rendimiento llamadas tiempo

de acceso y tiempo de sintonizado se usan para medir la eficiencia del acceso y la conservación de la potencia en sistemas de broadcast, respectivamente, y se definen de la siguiente manera:

- **Tiempo de acceso:** El tiempo que pasa entre el momento que una solicitud se realiza y el momento en que esta solicitud se satisface.

- **Tiempo de sintonizado:** El tiempo en el cual el cliente permanece activo para recibir la información solicitada.

Obviamente, el transmitir elementos de información irrelevantes aumenta el tiempo de acceso de los clientes, y como consecuencia, deteriora la eficiencia de un sistema de broadcast. Un programa de broadcast o programa de transmisión de archivos, el cual *determina que es lo que* debe ser transmitido por el servidor y cuando, debe ser diseñado de una forma muy cuidadosa. Existen tres tipos de modelos de transmisión por broadcast llamados: broadcast basados en empuje, broadcasts sobre demanda (o basados en jalar), y broadcasts híbridos. En los broadcasts basados en empuje, el servidor disemina la *información usando una programación periódica* o aperiódica (generalmente sin ninguna intervención de los clientes); en los broadcasts sobre demanda, el servidor disemina la información basándose en las solicitudes que envían los clientes; en los broadcasts híbridos, los basados en empuje y los broadcasts sobre demanda son combinados para complementarse uno al otro. *Como consecuencia existen tres tipos* de programación de la información (programación basada en empuje, sobre demanda, e híbrida).

En un sistema de broadcast, para obtener un elemento de información el cliente móvil debe de estar monitoreando continuamente *la transmisión hasta que el elemento* de información requerido llegue, esto consumirá gran parte de la potencia de la batería ya que el cliente debe permanecer activo durante su tiempo de espera.

2.2 Empujando información dependiente en sistemas Clientes–Proveedores–Servidores

En redes inalámbricas y satelitales y en sistemas avanzados de información de tráfico en donde el ancho de banda de enlace de subida es muy limitado, un servidor realiza un broadcast de archivos de información en forma cíclica. Los archivos de información son distribuidos por diferentes proveedores y son accesados por muchos clientes. Los proveedores son independientes y por lo tanto algunos archivos pueden compartir información. Los clientes que accesan estos archivos pueden tener diferentes patrones de acceso. Algunos clientes pueden desear accesar mas de un archivo en cualquier orden, algunos pueden accesar un archivo de varios existentes, y algunos clientes pueden accesar un segundo archivo solo después de accesar otro archivo. El objetivo del servidor es de ordenar los archivos de tal forma que se minimice el tiempo de acceso de los clientes dado algún conocimiento a priori de sus patrones de acceso.

2.2.1 Introducción

Muchas nuevas aplicaciones de redes tienen la propiedad que el canal de comunicación entre el servidor y el cliente es asimétrico. Esto siginifica que hay mucho más ancho de banda desde el servidor al cliente que en la dirección contraria. Por lo que, la información es diseminada (empujada) desde un servidor a muchos clientes en contraste con los métodos tradicionales en donde la información se jala desde los servidores hacia los clientes conforme se hacen las solicitudes.

Algunos ejemplos son:

1. Una red satelital en donde los satélites realizan un broadcast de las páginas web más populares, para dar mayor velocidad a la navegación del usuario. Cabe hacer notar que la programación del broadcast se basa en cierto conocimiento a priori de los patrones de acceso de los clientes.

2. Una red inalámbrica en donde las estaciones base pueden transmitir a razones mucho mayores que las que los clientes pueden transmitir. Se espera que en un futuro millones de usuarios móviles, con estaciones tipo palm-top, tendrán bastante demanda de distintos tipos de información.

3. Sistemas Avanzados de Información de Tráfico (Advanced Traffic Information Systems, ATIS) en donde cientos de miles de automovilistas simultáneamente puedan requerir información del servidor.

4. Televisión por cable o servicios de video en demanda con gran ancho de banda de bajada (servidor al cliente) y con un ancho de banda muy limitado de subida (cliente al servidor).

Una solución propuesta para canales asimétricos de comunicación es el usar sistemas de empuje (push systems). La idea central es que el servidor explote su ventaja en ancho de banda realizando broadcasts de información a múltiples clientes. En este caso, un servidor continuamente y repetidamente transmite información a una comunidad de clientes. En efecto, el canal de transmisión se convierte en un "disco" en el cual los clientes pueden tomar la información conforme pasa.

En el modelo tradicional para sistemas de empuje (push systems), existen dos protagonistas principales: los clientes y los servidores. Los servidores realizan broadcasts de los archivos de información y los clientes escuchan el canal de broadcast hasta que accesan la información que están buscando. El objetivo de los servidores es el de crear una programación que minimice el tiempo promedio de acceso de los clientes, y por medio de esto atraer clientes para futuros accesos.

Sin embargo, este sistema tradicional de clientes-servidores falla al tratar de cubrir todos los casos posibles. Hay muchos sistemas en el que los servidores no tienen la libertad de escoger los archivos de información. Estos solo dan sus servicios a otros proveedores los cuales necesitan realizar broadcasts. Es por esta razón que se utilizará el modelo de clientes-proveedores-servidores. En este modelo el servidor empuja los archivos de información a los clientes, pero no eligen los archivos a transmitir. En vez de eso, estos venden espacios de broadcast a proveedores que desean esparcir su información. Los proveedores seleccionan los servidores más exitosos y populares y los clientes seleccionan los servidores que transmiten la información que desean.

En el modelo tradicional en donde los servidores son los proveedores, estos podían decidir con que frecuencia transmitir los archivos. Este no es el caso en nuestro modelo. Aquí, la única flexibilidad dada a los servidores es la de fijar el orden en la que se transmitirán los archivos, esto se puede realizar debido a que en el contrato entre el servidor y el proveedor solo se especifica la razón a la que debe transmitirse un archivo. Nótese que se asumió que la programación es una permutación. Si el servidor no tiene algún conocimiento a priori acerca de los archivos más frecuentemente solicitados por los clientes, no puede hacer algo mejor que escoger una permutación arbitraria para transmitir los archivos. Esto se cumple aún cuando el servidor sabe la popularidad de cada archivo, puesto que no puede transmitir los archivos más populares con mayor frecuencia.

Si se conoce la distribución a priori del acceso de los clientes a *pares* de archivos se puede mejorar el desempeño del servidor. Se manejan tres variantes. En la primera variante, los clientes desean accesar ya sea uno o dos archivos en cualquier orden. En la segunda variante, los archivos pueden compartir información y los clientes pueden encontrar su información en cualquiera de los dos archivos dados. En la tercera variante, los clientes desean accesar dos archivos en cierto orden.

2.2.2 Definiciones

Sea $F=\{F_0, F_1, ..., F_{n-1}\}$ una colección de n archivos de igual tamaño. Se asume que toma una unidad de tiempo (un slot) para transmitir cada archivo. A continuación se definen las tres posibles probabilidades para los tres diferentes problemas que se consideran:

□ En el problema **AND**, para $0 \le i \le j \le n-1$, la probabilidad de accesar F_i y F_j es a_{ij}.

La probabilidad de accesar solo F_i es a_{ii}. Existen $\frac{n(n+1)}{2}$ diferentes probabilidades y $\sum_{j=0}^{n-1}\sum_{i=0}^{i\le j} a_{ij} = 1$ o de forma más general $\Sigma_{0 \le i \le j \le n-1} a_{ij} = 1$.

□ En el problema **OR**, para $0 \le i \le j \le n-1$, la probabilidad de accesar F_i o F_j es b_{ij}.

La probabilidad de accesar solo F_i es b_{ii}. Existen $\frac{n(n+1)}{2}$ diferentes probabilidades y $\sum_{j=0}^{n-1}\sum_{i=0}^{i\le j} b_{ij} = 1$ o de forma más general $\Sigma_{0 \le i \le j \le n-1} b_{ij} = 1$.

□ En el problema **IMPLY**, para $0 \le i \ne j \le n-1$, la probabilidad de accesar F_i y después F_j es c_{ij}. La probabilidad de accesar solo F_i es c_{ii}. Existen n^2 diferentes probabilidades y $\sum_{j=0}^{n-1}\sum_{i=0}^{n-1} c_{ij} = 1$ o de forma más general $\Sigma_{0 \le i, j \le n-1} c_{ij} = 1$.

La programación de un ciclo infinito se define como la permutación $\pi = (\pi(0), \pi(1), ..., \pi(n-1))$ en los números (0, 1, ..., n-1). En el tiempo $t \ge 0$ el servidor transmite el archivo $F_{\pi(i)}$ por una unidad de tiempo en donde $i = (t \bmod n)$. Sea $\sigma = \pi^{-1}$, es decir, $\sigma(\pi(i)) = i$ para toda $0 \le i \le n-1$. Decimos que los archivos F_i y F_j están a una distancia d_{ij} si:

$$d_{ij} = \min\{((\sigma(i) - \sigma(j)) \bmod n), ((\sigma(j) - \sigma(i)) \bmod n)\}$$

para $0 \leq i < j \leq n-1$. En otras palabras, d_{ij} es el ciclo más corto de distancia entre i y j en la permutación π y por lo tanto $d_{ij} \leq \frac{n}{2}$. Nótese que la distancia es una función simétrica, es decir, $d_{ij} = d_{ji}$ para $0 \leq i \leq j \leq n-1$.

Se dice que los archivos Fi y Fj están a una distancia directa si:

$$dd_{ij} = ((\sigma(j) - \sigma(i)) \bmod n)$$

para $0 \leq i, j \leq n-1$. En otras palabras, dd_{ij} es el número de slots entre el broadcast de F_i y el siguiente broadcast de F_j. Además $0 \leq dd_{ij} \leq n-1$. Nótese que la distancia directa es una función no simétrica, y que $dd_{ij} + dd_{ji} = n$ para $0 \leq i \neq j \leq n-1$.

Se asumió que los clientes llegan en slots de tiempo aleatorios (distribuidos uniformemente) durante la transmisión de la programación infinita. También se asumió que un cliente que desea accesar un archivo debe llegar antes del tiempo de inicio de transmisión del archivo. Si el cliente llega a la mitad de la transmisión de este archivo, debe esperar hasta que este archivo sea transmitido completamente.

Para una permutación π, sea $A_{ij}(\pi)$ el tiempo esperado de espera para un cliente que desea accesar los archivos F_i y F_j, sea $B_{ij}(\pi)$ el tiempo esperado de espera para un cliente que desea accesar ya sea el archivo F_i o el archivo F_j, y sea $C_{ij}(\pi)$ el tiempo esperado de espera para un cliente que desea accesar primero el archivo F_i y solo después el F_j. Sea

$$A(\pi) = \Sigma_{0 \leq i \leq j \leq n-1}[(a_{ij})(A_{ij}(\pi))]$$

$$B(\pi) = \Sigma_{0 \leq i \leq j \leq n-1}[(b_{ij})(B_{ij}(\pi))]$$

$$C(\pi) = \Sigma_{0 \leq i \neq j \leq n-1}[(c_{ij})(C_{ij}(\pi))]$$

Esto es, $A(\pi)$ es el tiempo esperado para todos los clientes que desean accesar uno o dos archivos, $B(\pi)$ es el tiempo esperado para todos los clientes que desean accesar un archivo o uno de dos archivos, y $C(\pi)$ es el tiempo esperado para todos los clientes que desean accesar un archivo o dos archivos en cierto orden.

En el problema AND se busca una permutación π que minimice $A(\pi)$, en el problema OR se busca una permutación π que minimice $B(\pi)$, y en el problema IMPLY se busca una permutación π que minimice $C(\pi)$.

2.2.3 Desarrollo

Para una permutación π dada y dos archivos F_i y F_j se muestra como calcular $A_{ij}(\pi)$, $B_{ij}(\pi)$, y $C_{ij}(\pi)$. Basándome en estos valores se muestra como calcular $A(\pi)$, $B(\pi)$, y $C(\pi)$.

Primero se verá el caso en el que i = j. En este caso, el cliente desea accesar solo un archivo. Nótese por las suposiciones hechas que los clientes deben accesar un archivo de comienzo a final y no pueden recordar el índice de un archivo. Como resultado, si un cliente llega poco después de haber iniciado la transmisión de F_i este debe esperar n slots de tiempo hasta la siguiente transmisión. Como el tiempo de llegada es aleatorio, en promedio el cliente espera $\frac{n}{2}$ slots hasta el inicio de la transmisión del slot de F_i, por lo tanto espera un slot más para completar el broadcast. Es decir,

$$A_{ii}(\pi) = B_{ii}(\pi) = C_{ii}(\pi) = \frac{n}{2} + 1 \qquad \text{.....(2.1)}$$

Ahora se verá el caso en el que $i \neq j$ y que la distancia entre F_i y F_j en la permutación π es $d = d_{ij}$ para los problemas AND y OR y es $dd = dd_{ij}$ para el problema IMPLY. El valor de d es por lo menos 1 y cuando mucho $\frac{n}{2}$ y el valor de dd es por lo menos 1 y cuando mucho n-1.

Se empezará calculando $C_{ij}(\pi)$. Por la ecuación (2.1), el tiempo promedio de acceso para el primer archivo F_i es $\frac{n}{2} + 1$ slots. Recuérdese que en el problema IMPLY el cliente ignora F_j si todavía no ha accesado F_i. Por lo tanto cualquier cliente tiene que esperar dd slots adicionales para F_j independientemente de su punto de llegada. Por lo que,

$$C_{ij}(\pi) = \frac{n}{2} + 1 + dd \qquad \text{.....(2.2)}$$

Nótese que para i = j, la ecuación (2.2) cumple con la ecuación (2.1) ya que dd = 0 en este caso. Como resultado,

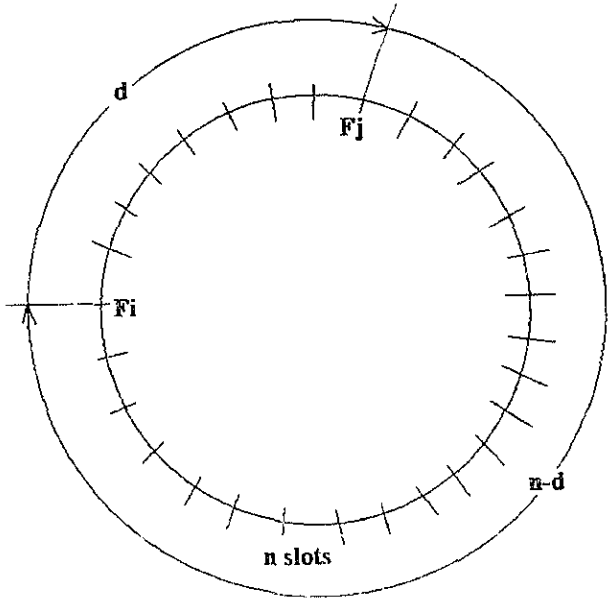
$$C(\pi) = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} c_{ij} C_{ij}(\pi)$$

$$C(\pi)=\sum_{j=0}^{n-1}\sum_{i=0}^{n-1}c_{ij}(\frac{n}{2}+1+dd_{ij})$$

$$\boxed{C(\pi)=\frac{n}{2}+1+\sum_{j=0}^{n-1}\sum_{i=0}^{n-1}c_{ij}(dd_{ij})} \quad \text{.....(2.3)}$$

Se llegó a la ecuación (2.3) dado que $\sum_{j=0}^{n-1}\sum_{i=0}^{n-1}c_{ij}=1$.

Ahora se calculará $A_{ij}(\pi)$. Asuma que la distancia d es de F_i a F_j. Por lo tanto, como la programación es cíclica, la distancia desde F_j hasta F_i es n-d. Con esto se distinguen dos intervalos de llegada. El primer intervalo de distancia d es desde F_i hasta F_j y el segundo intervalo de distancia n-d es desde F_j hasta F_i. Un cliente que llega durante el primer intervalo espera en promedio $\frac{d}{2}$ slots de tiempo antes de la transmisión de F_j. Después de eso espera exactamente n-d+1 slots de tiempo para accesar ambos archivos. Un cliente que llega durante el segundo intervalo espera en promedio $\frac{n-d}{2}$ slots antes de la transmisión de F_i. Después de eso espera exactamente d+1 slots para accesar ambos archivos. Para hacer más clara la explicación obsérvese la siguiente figura:



Por lo que el tiempo de acceso de acuerdo con las longitudes de los intervalos es:

$$A_{i,j}(\pi) = \frac{d}{n}\left(\frac{d}{2} + (n-d+1)\right) + \frac{n-d}{n}\left(\frac{n-d}{2} + (d+1)\right)$$

$$\boxed{A_{i,j}(\pi) = \left(\frac{n}{2}+1\right) + \left(d - \frac{d^2}{n}\right)} \quad \text{.....(2.4)}$$

Nótese que para el caso en el que i = j, la ecuación (2.4) cumple con la ecuación (2.1) puesto que d = 0 para este caso. Nótese también que $A_{ij}(\pi)$ es más pequeño que $C_{ij}(\pi)$ ya que los clientes pueden accesar ambos archivos en cualquier orden. Como resultado,

$$A(\pi) = \Sigma_{0 \le i \le j \le n-1}[(a_{ij})(A_{ij}(\pi))]$$

$$A(\pi) = \sum_{j=0}^{n-1}\sum_{i=0}^{i \le j}[a_{i,j}A_{i,j}(\pi)]$$

$$A(\pi) = \sum_{j=0}^{n-1}\sum_{i=0}^{i \le j} a_{i,j}\left(\left(\frac{n}{2}+1\right) + \left(d_{i,j} - \frac{d_{i,j}^{\,2}}{n}\right)\right)$$

$$\boxed{A(\pi) = \frac{n}{2} + 1 + \sum_{j=0}^{n-1}\sum_{i=0}^{i \le j} a_{ij}\left(d_{i,j} - \frac{d_{i,j}^{\,2}}{n}\right)} \quad \text{.....(2.5)}$$

La última ecuación se obtuvo ya que $\sum_{j=0}^{n-1}\sum_{i=0}^{i \le j} a_{i,j} = 1$

Ahora se calculará $B_{ij}(\pi)$. Una vez más se asume que la distancia d es desde F_i a F_j y la de F_j a F_i es n-d. Se distinguen dos intervalos de llegada. El primer intervalo de longitud d es desde F_i a F_j y el segundo intervalo de distancia n-d es desde F_j a F_i. Un cliente que llega durante el primer intervalo espera en promedio $\frac{d}{2}$ slots de tiempo antes de la transmisión de F_j. Después de eso espera un slot adicional para accesar F_j. Un cliente

que llega durante el segundo intervalo espera en promedio $\frac{n-d}{2}$ slots antes de la transmisión de F_i. Después de eso espera un slot adicional para accesar F_i.

Por lo tanto el tiempo de acceso de acuerdo con las longitudes de los intervalos es:

$$B_{ij}(\pi) = \frac{d}{n}\left(\frac{d}{2}+1\right)+\frac{n-d}{n}\left(\frac{n-d}{2}+1\right)$$

$$\boxed{B_{ij}(\pi)=\left(\frac{n}{2}+1\right)-\left(d-\frac{d^2}{n}\right)} \quad \text{.....(2.6)}$$

Nótese que para el caso en el que i = j, la ecuación (2.6) concuerda con la ecuación (2.1) ya que d = 0 para este caso. También nótese que $B_{ij}(\pi)$ es más pequeño que $A_{ij}(\pi)$ ya que los clientes solo necesitan accesar uno de los dos archivos de F_i y F_j. Como resultado,

$$B(\pi) = \Sigma_{0 \le i \le j \le n-1}[(b_{ij})(B_{ij}(\pi))]$$

$$B(\pi) = \sum_{j=0}^{n-1}\sum_{i=0}^{i\le j}[b_{ij}B_{ij}(\pi)]$$

$$B(\pi) = \sum_{j=0}^{n-1}\sum_{i=0}^{i\le j} b_{ij}\left[\left(\frac{n}{2}+1\right)-\left(d_{ij}-\frac{d_{ij}^2}{n}\right)\right]$$

$$\boxed{B(\pi) = \frac{n}{2}+1-\sum_{j=0}^{n-1}\sum_{i=0}^{i\le j} b_{ij}\left(d_{ij}-\frac{d_{ij}^2}{n}\right)} \quad \text{.....(2.7)}$$

La última ecuación se obtuvo ya que $\sum_{j=0}^{n-1}\sum_{i=0}^{i\le j} b_{ij} = 1$.

2.3 El Algoritmo de Hill Climbing.

Este algoritmo es una técnica heurística. Heurístico significa métodos, criterios o principios para decidir entre varios caminos alternos de decisiones o acciones usadas para determinar soluciones que toman tiempo infinito o tiempo muy extenso en procesamiento

computacional. Estos métodos no siempre encuentran la mejor solución, pero garantizan encontrar una muy buena en un tiempo razonable.

Antes de mostrar este algoritmo, se darán ciertos conceptos y definiciones necesarios para comprender claramente el ya mencionado algoritmo.

2.3.1 Algunos operadores para permutaciones

Existen la selección, el cruce, y la mutación, en este caso solo explicaré la mutación pues es el operador que se utilizó además de que no tiene caso el explicar los otros operadores puesto que no es el objetivo de la presente tesis.

- Mutación: Este operador reordena los elementos de una permutación elegidos de forma aleatoria, algunos tipos de mutación son:

a. Intercambio
Esta mutación elige dos elementos y los intercambia. Por ejemplo, en la permutación 1234567, si los elementos 2 y 6 se eligen, entonces la nueva permutación sería 1634527. Este tipo de mutación es la usada en esta tesis.

b. Mover un bloque
Esta mutación elige un bloque aleatorio de elementos y los mueve a un punto dentro de la permutación. Por ejemplo, si la permutación es 123456, el bloque es 345 y la posición es 1, entonces la nueva permutación sería 345126.

c. Intercambio de bloques
Esta mutación elige dos bloques y los intercambia. Por ejemplo para la permutación 12345678, si los bloques son 23 y 67 entonces la nueva permutación sería 16745238.

d. Inserción
Elige un elemento y lo inserta en otra posición en la permutación. Por ejemplo, si la permutación es 2345678 y el número 3 se elige con la posición 6 la nueva permutación sería 2456738

e. 2-opt
Elige 2 puntos a lo largo de una cadena, luego invierte el segmento entre los dos puntos. Por ejemplo, si la permutación es 12345, y los dos puntos son 2 y 4, entonces la nueva permutación sería 14325.

f. Mezcla
Elige dos puntos, y de forma aleatoria los reordena. Por ejemplo si la permutación es 12345 y los puntos 2 y 4, la nueva permutación podría reordenarse como 13425.

2.3.2 Hill Climbing

La técnica de hill climbing sirve para evaluar una solución existente y luego moverse a una mejor solución que se acerque más al objetivo. Para determinar si una solución es mejor que otra, se necesita un tipo de evaluación o función. La solución se alcanza cuando la función a evaluar es mínima. La función de evaluación es comúnmente llamada función de propiedad. En el caso del presente trabajo las funciones de propiedad son las ecuaciones 2.3, 2.5, y 2.7.

Primero se define una solución inicial, digamos So, y se calcula la propiedad de So. Lo siguiente es definir uno, algunos o todos los vecinos de So. Un vecino de una permutación se obtiene aplicando cualquiera de los operadores de permutaciones vistos anteriormente. Se tienen dos instancias del algoritmo de hill climbing.

En la primera instancia, se genera una muestra de vecinos alrededor de So. Se calcula la propiedad de todos los vecinos y el mejor vecino S´ se determina. Se realiza un movimiento de So a S´. En la siguiente iteración, se genera una muestra de los vecinos de S´. Este método iterativo continua hasta que no se encuentra una mejor solución. El algoritmo se describe a continuación:

```
Elegir una solución inicial s en X
alto := falso
mientras alto = falso hacer
        Generar una muestra V* de soluciones en N(s) {N(s) es la vecindad de s}
Encontrar la mejor s´ en V* {evaluar cada vecino y encontrar s´ tal que la
                            propiedad de s´ sea mínima entre todos los vecinos}
si Propiedad(s´) >= Propiedad(s)
  entonces alto := verdadero
  otro s := s´
fin de mientras
```

En la segunda instancia, un solo vecino s´ es generado en cada iteración y se calcula su propiedad. Si s´ es mejor que s_o, se convierte en la permutación candidata (se hace un movimiento de s_o a s´), y en la siguiente iteración se calcula un vecino de s´. Si s´ no es mejor que s_o, entonces s´ se ignora y se genera otro vecino de s_o. El proceso se detiene hasta que no se obtiene mejoría alguna. El algoritmo se describe a continuación:

```
So := solución inicial;
MejorSolución := So;
Propiedad1 := P(So);
i := 0;
repetir
  C := siguiente vecino (Si) {un solo vecino}
  Propiedad2 := P(C);
  si Propiedad2 < Propiedad1
```

```
        entonces
                inicio
                        MejorSolución := C;
                        S_{i+1} := C;
                        i := i+1;
                        Propiedad1 := Propiedad2;
                fin
        otro
                inicio
                        S_{i+1} := Si;
                        i := i+1;
                fin
hasta que i = max Ó si Si no tiene más vecinos
```

Estos procedimientos son usados para buscar una solución óptima en un problema de optimización, pero no se garantiza que exista un mínimo global, es decir, esta técnica sirve muy bien para encontrar mínimos locales pero no se garantiza que el mínimo local sea el mínimo global del problema.

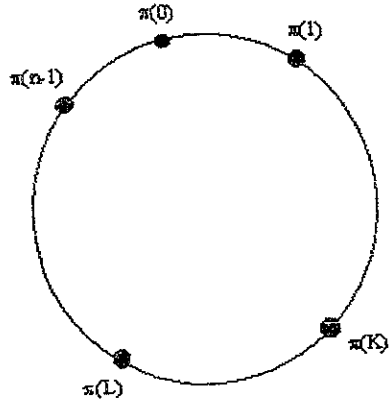
Capítulo III

En este capítulo se mostrará el desarrollo y la explicación de la base de la solución implementada.

3.1 La evolución de las permutaciones por medio del operador intercambio y su relación con las matrices de distancias d_{ij} y distancias directas dd_{ij}.

Una permutación π es un ciclo de archivos ordenados de cierta forma, esto es:

$\pi = \{\pi(0), \pi(1), \pi(2), \ldots, \pi(n-1)\}$



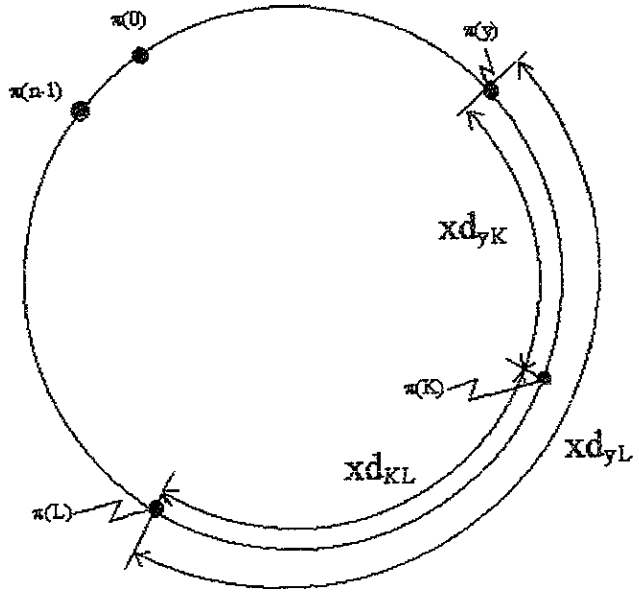
Como no podemos repetir los archivos que son más populares, lo único que podemos hacer es reordenar la permutación π para minimizar $A(\pi)$, $B(\pi)$ y $C(\pi)$, si se observan las ecuaciones (2.3), (2.5) y (2.7) se nota que a_{ij}, b_{ij} y c_{ij} son dados, n también es dado y constante por lo que lo único que se puede cambiar son las distancias de los archivos, esto es, reordenar los archivos de la permutación π:

- Se calcula $X(\pi)$ (X puede ser A, B o C, según sea el caso, recordándose que A=AND, B=OR y C=IMPLY).
- Se aplica un operador evolutivo a la permutación π para obtener una nueva permutación π', en este caso se aplica el operador de **intercambio o swap.**
- Se comparan $X(\pi')$ con $X(\pi)$ y se elige la mejor es decir, la que tenga menor propiedad.
- Estos pasos se repiten hasta que no se encuentra mejora alguna.

Para calcular $X(\pi)$ se necesitan x_{ij} y xd_{ij}, donde x_{ij} es la matriz de probabilidades dependientes entre los pares de archivos i y j, y xd_{ij} es la matriz de distancias o de distancias directas entre los archivos i y j.

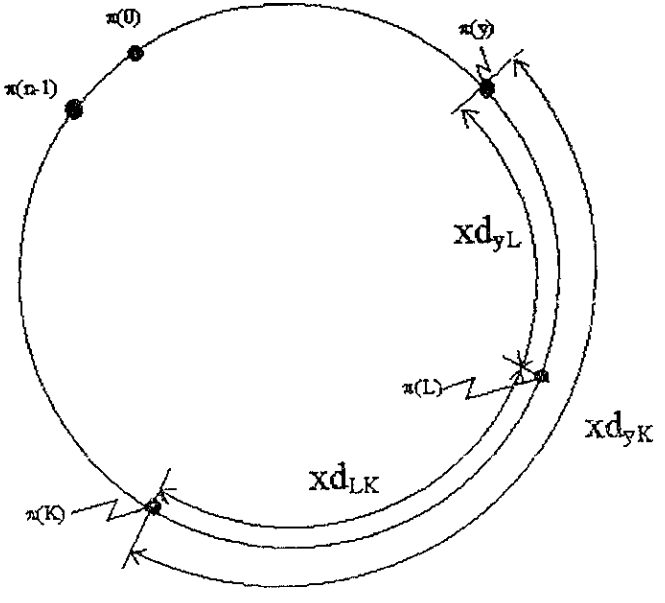
Para minimizar los cálculos de permutación a permutación al momento de realizar el intercambio de archivos se hizo lo siguiente:

Figura 1. π:



Si se intercambian los archivos K y L, es decir, si se aplica el operador intercambio se *obtiene* la Figura 2.

π':



Observando las figuras 1 y 2 se nota que al momento de aplicar el operador intercambia a los elementos L y K de la permutación π se llega a las siguientes relaciones:

xd_{yK} cambia a xd_{yL}
xd_{KL} cambia a xd_{LK}
xd_{yL} cambia a xd_{yK}

Viendo estas relaciones se concluyó que se debía intercambiar los renglones y las columnas de los archivos Ky L en la matriz xd_{ij} para obtener la nueva matriz de distancias, este paso fue trascendente puesto que en lugar de calcular toda la nueva matriz de distancias solo se hace un reordenamiento en la matriz.

Mostrando la matriz xd_{ij}.

$$xdij = \begin{bmatrix} 0 & xd_{0,1} & \cdots & xd_{0,y} & \cdots & xd_{0,K} & \cdots & xd_{0,L} & \cdots & xd_{0,n-1} \\ xd_{1,0} & 0 & \cdots & xd_{1,y} & \cdots & xd_{1,K} & \cdots & xd_{1,L} & \cdots & xd_{1,n-1} \\ \vdots & \vdots & \cdots & \vdots & \cdots & \vdots & \cdots & \vdots & \cdots & \vdots \\ xd_{y,0} & xd_{y,1} & \cdots & 0 & \cdots & xd_{y,K} & \cdots & xd_{y,L} & \cdots & xd_{y,n-1} \\ \vdots & \vdots & \cdots & \vdots & \cdots & \vdots & \cdots & \vdots & \cdots & \vdots \\ xd_{K,0} & xd_{K,1} & \cdots & xd_{K,y} & \cdots & 0 & \cdots & xd_{K,L} & \cdots & xd_{K,n-1} \\ \vdots & \vdots & \cdots & \vdots & \cdots & \vdots & \cdots & \vdots & \cdots & \vdots \\ xd_{L,0} & xd_{L,1} & \cdots & xd_{L,y} & \cdots & xd_{L,K} & \cdots & 0 & \cdots & xd_{L,n-1} \\ \vdots & \vdots & \cdots & \vdots & \cdots & \vdots & \cdots & \vdots & \cdots & \vdots \\ xd_{n-1,0} & xd_{n-1,1} & \cdots & xd_{n-1,y} & \cdots & xd_{n-1,K} & \cdots & xd_{n-1,L} & \cdots & 0 \end{bmatrix}$$

Nótese que $xd_{ij} = 0$.

Por medio del intercambio de las columnas y los renglones de la matriz xd_{ij} no se tiene que calcular toda la nueva matriz d_{ij}' para la permutación π'.

Análisis para el problema AND

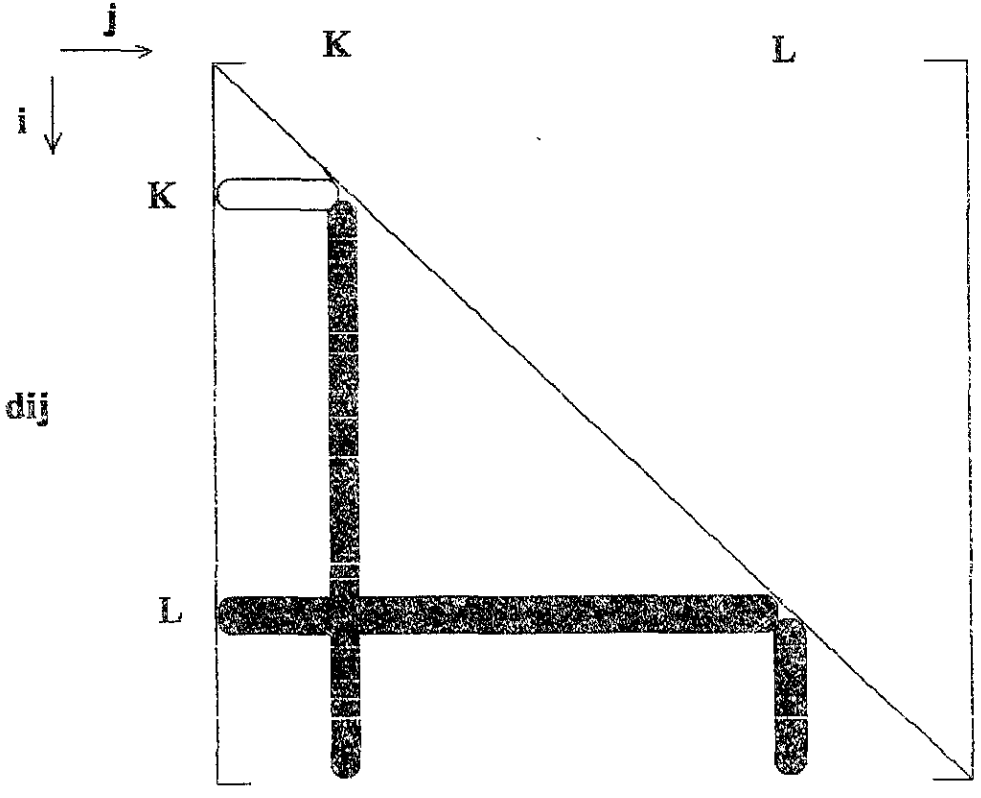
$$A(\pi) = \frac{n}{2} + 1 + \sum_{j=0}^{n-1} \sum_{i=0}^{i \le j} a_{ij} (d_{ij} - \frac{d_{ij}^{\,2}}{n})$$

$$A(\pi') = A(\pi) - suma1 + suma2$$

$$suma1 = \sum_{j=0}^{K-1} a_{Kj} (d_{Kj} - \frac{d_{Kj}^{\,2}}{n}) + \sum_{j=0}^{L-1} a_{Lj} (d_{Lj} - \frac{d_{Lj}^{\,2}}{n}) + \sum_{i=K+1}^{n-1} a_{iK} (d_{iK} - \frac{d_{iK}^{\,2}}{n}) + \sum_{i=L+1}^{n-1} a_{iL} (d_{iL} - \frac{d_{iL}^{\,2}}{n})$$

$$- a_{LK} (d_{LK} - \frac{d_{LK}^{\,2}}{n})$$

Si se observa la siguiente figura se entiende la forma en que se van haciendo las sumas:



$$suma2 = \sum_{j=0}^{K-1} a_{Kj}(d'_{Kj} - \frac{d'^{2}_{Kj}}{n}) + \sum_{j=0}^{L-1} a_{Lj}(d'_{Lj} - \frac{d'^{2}_{Lj}}{n}) + \sum_{i=K+1}^{n-1} a_{iK}(d'_{iK} - \frac{d'^{2}_{iK}}{n}) + \sum_{i=L+1}^{n-1} a_{iL}(d'_{iL} - \frac{d'^{2}_{iL}}{n})$$

$$-a_{LK}(d'_{LK} - \frac{d'^{2}_{LK}}{n})$$

La suma2 se obtiene de la misma forma que suma1 solo que ahora se utiliza la nueva matriz de distancias d_{ij}' que se obtuvo intercambiando las columnas y los renglones K y L, debo hacer notar que para el cálculo de suma 2 se sigue utilizando la misma matriz de probabilidades a_{ij}.

Para el problema OR.

$$B(\pi) = \frac{n}{2} + 1 - \sum_{j=0}^{n-1} \sum_{i=0}^{i \le j} b_{ij}(d_{ij} - \frac{d_{ij}^{2}}{n})$$

$$B(\pi') = B(\pi) + suma1 - suma2$$

$$suma1 = \sum_{j=0}^{K-1} b_{Kj}(d_{Kj} - \frac{d_{Kj}^{2}}{n}) + \sum_{j=0}^{L-1} b_{Lj}(d_{Lj} - \frac{d_{Lj}^{2}}{n}) + \sum_{i=K+1}^{n-1} b_{iK}(d_{iK} - \frac{d_{iK}^{2}}{n}) + \sum_{i=L+1}^{n-1} b_{iL}(d_{iL} - \frac{d_{iL}^{2}}{n})$$

$$-b_{LK}(d_{LK} - \frac{d_{LK}^{2}}{n})$$

$$suma2 = \sum_{j=0}^{K-1} b_{Kj}(d'_{Kj} - \frac{d'^{2}_{Kj}}{n}) + \sum_{j=0}^{L-1} b_{Lj}(d'_{Lj} - \frac{d'^{2}_{Lj}}{n}) + \sum_{i=K+1}^{n-1} b_{iK}(d'_{iK} - \frac{d'^{2}_{iK}}{n}) + \sum_{i=L+1}^{n-1} b_{iL}(d'_{iL} - \frac{d'^{2}_{iL}}{n})$$

$$-b_{LK}(d'_{LK} - \frac{d'^{2}_{LK}}{n})$$

En si para el problema OR el desarrollo de las sumas es igual al del problema AND solo cambia el signo de las sumas.

ara el problema IMPLY

$$(\pi) = \frac{n}{2} + 1 + \sum_{j=0}^{n-1}\sum_{i=0}^{n-1} c_{i,j}(dd_{i,j})$$

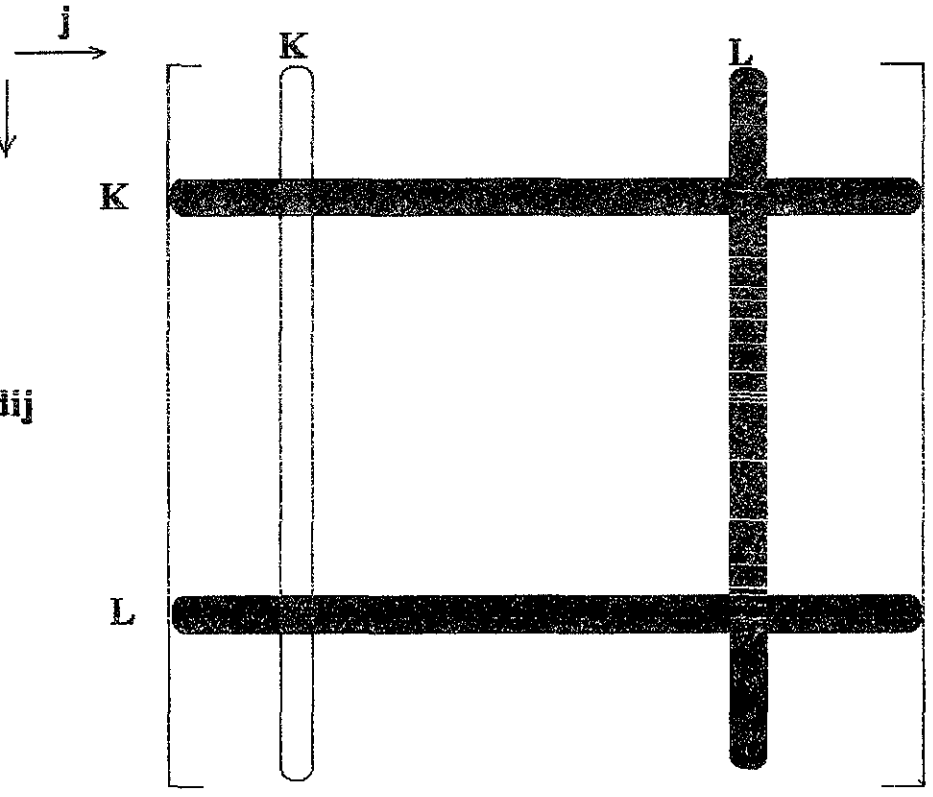
$$(\pi') = C(\pi) - suma1 + suma2$$

$$uma1 = \sum_{j=0}^{n-1} b_{K,j}(dd_{K,j} - \frac{dd_{K,j}^{\ 2}}{n}) + \sum_{j=0}^{n-1} b_{L,j}(dd_{L,j} - \frac{dd_{L,j}^{\ 2}}{n}) + \sum_{i=o}^{n-1} b_{i,K}(dd_{i,K} - \frac{dd_{i,K}^{\ 2}}{n}) + \sum_{i=0}^{n-1} b_{i,L}(dd_{i,L} - \frac{dd_{i,L}^{\ 2}}{n})$$
$$b_{KK}(dd_{KK} - \frac{dd_{KK}^{\ 2}}{n}) - b_{LK}(dd_{LK} - \frac{dd_{LK}^{\ 2}}{n}) - b_{KL}(dd_{KL} - \frac{dd_{KL}^{\ 2}}{n}) - b_{LL}(dd_{LL} - \frac{dd_{LL}^{\ 2}}{n})$$

$$uma2 = \sum_{j=0}^{n-1} b_{K,j}(dd'_{K,j} - \frac{dd'_{K,j}^{\ 2}}{n}) + \sum_{j=0}^{n-1} b_{L,j}(dd'_{L,j} - \frac{dd'_{L,j}^{\ 2}}{n}) + \sum_{i=o}^{n-1} b_{i,K}(dd'_{i,K} - \frac{dd'_{i,K}^{\ 2}}{n}) + \sum_{i=0}^{n-1} b_{i,L}(dd'_{i,L} - \frac{dd'_{i,L}^{\ 2}}{n})$$
$$b_{KK}(dd'_{KK} - \frac{dd'_{KK}^{\ 2}}{n}) - b_{LK}(dd'_{LK} - \frac{dd'_{LK}^{\ 2}}{n}) - b_{KL}(dd'_{KL} - \frac{dd'_{KL}^{\ 2}}{n}) - b_{LL}(dd'_{LL} - \frac{dd'_{LL}^{\ 2}}{n})$$

a suma2 se obtiene de la misma forma que suma1 solo que ahora se utiliza la nueva atriz de distancias d_{ij}' que se obtuvo intercambiando las columnas y los renglones K y , debo hacer notar que para el cálculo de suma2 se sigue utilizando la misma matriz de robabilidades c_{ij}.

En la siguiente figura se puede observar el porque la estructura de suma1 y suma2.



Creo que es evidente la ventaja de ir intercambiando los elementos K y L de las matrices d y dd según sea el caso, ya que en lugar de calcular la propiedad de las nuevas permutaciones π' con todos los elementos de d_{ij}' o dd_{ij}' solo se tienen que hacer las sumas y las restas de los renglones y las columnas de L y de K, esto implica una gran minimización en el tiempo de procesamiento ya que en vez de hacer en promedio nxn operaciones se realizan en promedio 2n operaciones, haciendo muchísimo más rápido la evolución entre permutación y permutación.

Capítulo IV

En el presente capítulo se muestra el análisis y la simulaciones de la solución implementada además de la comparación de los resultados ya existentes con los obtenidos en esta tesis.

4.1 Análisis y Simulación

Ya se ha hablado de forma extensa acerca del problema que se quiere resolver, de la teoría necesaria y del desarrollo de las herramientas matemáticas que se utilizarán, a continuación se mostrarán la solución de los tres casos propuestos AND, OR e IMPLY con el algoritmo de Hill Climbing al cual se le hizo una modificación inicial para mejorar su desempeño, la simulación se programó en la versión 5.3.1.29215a (R11.1) de Matlab. Se eligió Matlab debido a su sencillo uso y a que se tienen implementadas muchas funciones las cuales fueron de gran utilidad para realizar la simulación, es importante aclarar que si en un futuro se desea usar estos resultados para implementar el programa en una aplicación real, convendría más hacer un programa en un lenguaje como C o C++ para mejorar el tiempo de ejecución del programa.

4.1.1 Generación de las variables usadas en la simulación

- **Generación de las permutaciones π.**

Las permutaciones se generarán con la ayuda de la función de Matlab randperm, la cual genera una permutación aleatoria de tamaño n con números enteros entre 1 y n.

Ejemplo:
Para n = 4

» randperm (4)

ans =

2 4 3 1

en este caso se programó una función para que los valores de los elementos de la permutación π estuvieran entre cero y uno, además de que tuvieran sus respectivos subíndices, esto con el fin de facilitar el cálculo de su función inversa σ.

Ejemplo: para n = 5

» genera_pi(5)

ans =

0 1 2 3 4
3 2 1 4 0

- **Generación de las matrices de probabilidades a_{ij}, b_{ij} y c_{ij}.**

En este caso, para simular de manera más fiel un sistema real, dentro de la permutación de la programación a transmitir se supondrán grupos de archivos con temática en común, por ejemplo, si una programación consta de archivos de deportes, noticias, etc..., las probabilidades entre los archivos con temática similar serán mayores que las probabilidades entre archivos con temática diferente.

Teniendo n, es decir el número de archivos a transmitir o el tamaño de la permutación π, se hicieron grupos de archivos de forma arbitraria, por forma arbitraria quiero decir que se eligieron manualmente los grupos de archivos.

Si se tiene una permutación de 10 archivos se puede suponer que 3 archivos son de un tipo, 5 de otro tipo y 2 de otro, de esta forma las probabilidades dentro de cada grupo son mayores que las probabilidades entre archivos de diferentes grupos, esto se realizó con la intención de acercarse más a la realidad ya que una persona que busque archivos de un cierto tipo puede ser más probable que siga buscando archivos del mismo tipo que de otro tipo.

Como ejemplo, para el problema AND, si se tiene una programación de 10 archivos los cuales están divididos en tres grupos:

Grupo 1.- deportes (del archivo 1 al 4 en la permutación)
Grupo 2.- noticias (del archivo 5 al 7 en la permutación)
Grupo 3.- ciencia (del archivo 8 al 10 en la permutación)

Se pensó lógico que la probabilidad de accesar dos archivos del Grupo 1 tiene mayor probabilidad que la de accesar un archivo del Grupo 1 y un archivo del Grupo 2.

Para generar las matrices de probabilidades se programaron dos funciones:

xij = and_or (n, files) para los casos AND y OR

xij = imply (n, files) para el caso IMPLY

donde n es el tamaño de la permutación a transmitir y files es un arreglo que contiene los límites de los grupos de archivos, xij es la matriz de probabilidades a_{ij}, b_{ij} o c_{ij} según sea el caso, retomando el ejemplo pasado:

Grupo 1.- deportes (del archivo 1 al 4 en la permutación)
Grupo 2.- noticias (del archivo 5 al 7 en la permutación)
Grupo 3.- ciencia (del archivo 8 al 10 en la permutación)

El arreglo files es [4 7 10].

La función and_or realiza lo siguiente:

1. Crea una matriz "p" de nxn con valores aleatorios tomados de una distribución normal, esto lo hace con la función randn de Matlab, como esta función devuelve tanto valores positivos como negativos entre cero y uno, solo se obtuvo el valor absoluto de todos los valores para tener valores positivos entre cero y uno.

2. Crea una matriz "a" de tamaño nxn, en donde le asigna un valor de $1x10^{-6}$ si tiene coordenadas que se encuentren en el rango de combinación de grupos de archivos del arreglo files, en cambio asigna el valor de uno a las coordenadas que se encuentren en el rango del mismo grupo de archivos del arreglo files.

3. Realiza la multiplicación punto a punto de "p" y "a".

4. En este caso como $x_{ij} = x_{ji}$ a estas coordenadas las suma y las divide entre 2 asignando a las dos el resultado obtenido.

5. Hace la normalización para cumplir con:

$$\sum_{j=0}^{n-1}\sum_{i=0}^{i\leq j} x_{i,j} = 1$$

La función imply realiza lo siguiente:

6. Crea una matriz "p" de nxn con valores aleatorios tomados de una distribución normal, esto lo hace con la función randn de Matlab, como esta función devuelve tanto valores positivos como negativos entre cero y uno, solo se obtuvo el valor absoluto de todos los valores para tener valores positivos entre cero y uno.

7. Crea una matriz "a" de tamaño nxn, en donde le asigna un valor de $1x10^{-6}$ si tiene coordenadas que se encuentren en el rango de combinación de grupos de archivos del arreglo files, en cambio asigna el valor de uno a las coordenadas que se encuentren en el rango del mismo grupo de archivos del arreglo files.

8. Realiza la multiplicación punto a punto de "p" y "a".

9. Hace la normalización para cumplir con:

$$\sum_{i=0}^{n-1}\sum_{j=0}^{n-1} x_{i,j} = 1$$

A continuación se muestra un ejemplo de las matrices a_{ij}, b_{ij} y c_{ij} para n=5 y files = [2 3 5], debe recalcarse que algunos valores presentados en las matrices aunque parece que son cero, en realidad no lo son, son valores muy pequeños que Matlab no despliega pero que en realidad si almacena los valores para su posterior uso .

a_{ij} =

0.2103	0.1926	0.0000	0.0000	0.0000
0.1926	0.0760	0.0000	0.0000	0.0000
0.0000	0.0000	0.0207	0.0000	0.0000
0.0000	0.0000	0.0000	0.3913	0.0782
0.0000	0.0000	0.0000	0.0782	0.0309

b_{ij} =

0.2103	0.1926	0.0000	0.0000	0.0000
0.1926	0.0760	0.0000	0.0000	0.0000
0.0000	0.0000	0.0207	0.0000	0.0000
0.0000	0.0000	0.0000	0.3913	0.0782
0.0000	0.0000	0.0000	0.0782	0.0309

c_{ij} =

0.0084	0.0281	0.0000	0.0000	0.0000
0.0685	0.0711	0.0000	0.0000	0.0000
0.0000	0.0000	0.1038	0.0000	0.0000
0.0000	0.0000	0.0000	0.1921	0.2034
0.0000	0.0000	0.0000	0.1632	0.1614

Existen 3 grupos de archivos el primero va del archivo uno al dos, el segundo solo está formado por el archivo tres y el tercero del cuarto al quinto archivo.

4.1.2 Implementación del algoritmo de hill climbing.

Ya se mencionó que el algoritmo de hill climbing es una técnica eficiente para encontrar un mínimo local a partir de una solución inicial, en nuestro caso a partir de una permutación inicial, desgraciadamente no garantiza que el mínimo encontrado a partir de una permutación inicial sea el valor mínimo del problema global, para solucionar este problema se programó el algoritmo partiendo de varias permutaciones iniciales, es decir, en lugar de solo tomar una permutación inicial se generaron varias permutaciones inicialesal final se compararon los mínimos obtenidos después de aplicar el algoritmo a cada permutación inicial y se eligió la que entregó un menor valor, de esta forma podemos acercarnos más a un mínimo global del problema aunque de todas formas no se garantiza obtener el mínimo global. A esta modificación se le podría llamar como algoritmo de hill climbing paralelo.

En la implementación del algoritmo es muy importante el mencionar como se eligieron las permutaciones de siguientes vecinos, es decir, como se realizó la evolución entre permutación y permutación, además de decir cual fue el criterio a seguir para determinar que no se podía obtener una mejor solución.

Como se mencionó en el *Capíulo II se utilizó el operador mutación*, en específico el Intercambio o Swap de dos elementos de la permutación, la pregunta sería ¿cuál fue el criterio a seguir para realizar los intercambios?
El criterio para intercambiar los archivos K y L fue de forma aleatoria, es decir, teniendo la permutación π se generó una permutación auxilar aleatoria y se *tomaron los dos primeros valores como* los valores de L y de K para intercambiar los archivos de la permutación π, de esta forma cada vez que se necesitaba evolucionar a una nueva permutación se generaban de forma aleatoria los valores de L y de K.

En este caso particular se programó la segunda instancia del algortimo de hill climbing anteriormente explicada, en donde se parte de una pemutación inicial y se evoluciona a un solo vecino por iteración, comparando si la nueva permutación es mejor que su antecesora, tomando en cuenta como criterio de decisión para saber si se llega a un mínimo un número máximo de iteraciones por permutación, en sí se hizo lo siguiente:

Se generó un número t de permutaciones iniciales.
Para cada permutación So de t:

```
So := solución inicial;
MejorSolución := So;
Propiedad1 := P(So);
max := 0;
```

```
repetir
  C := siguiente vecino  {un solo vecino}
  Propiedad2 := P(C);
  si Propiedad2 < Propiedad1
        entonces
              inicio
                    MejorSolución := C;
                    Sc := C;
                    max := 0;
                    Propiedad1 := Propiedad2;
              fin
        otro
              inicio
                    max := max+1;
              fin
hasta que max < 30
                    (En este caso después de hacer varias pruebas elegí el valor
                    max = 30.)
```

Elegir la permutación C que tenga menor valor entre todas las obtenidas del conjunto t de permutaciones iniciales.

4.1.3 Primer simulación: Comparación de los resultados finales del algoritmo de hill climbing con los del algoritmo aleatorio.

Se realizó lo siguiente:

- Se creo un conjunto de t permutaciones aleatorias de tamaño n, con sus respectivos grupos de archivos files para generar las matrices de probabilidades.

- A cada permutación se le aplicó el algoritmo de hill climbing para mejorar la permutación; se contó el número total de permutaciones para las cuales se evaluó la función de propiedad. Se hizo que la cuenta_total fuese el número total de permutaciones evaluadas, para todas las permutaciones t aplicadas al algoritmo de hill climbing. Se eligió la mejor permutación P1.

- Se generó un conjunto cuenta_total de permutaciones aleatorias y se eligió la mejor permutación P2 entre todas.

- Se comparó P1 y P2, es decir, el tiempo de acceso y el tiempo de procesamiento computacional.

Todo esto se mostrará en una tabla comparativa, pero antes se mostrará un ejemplo de una permutación con su permutación inversa, su matriz de distancias y su matriz de distancias directas, esto para hacer más ilustrativo la forma en que funciona el programa en su conjunto.

Sea la permutación: (los cocficientes de arriba son el índice y abajo son los valores de los archivos)

p =

0	1	2	3	4	5	6	7	8
7	1	6	3	2	5	8	4	0

la permutación s inversa de p es:

s =

0	1	2	3	4	5	6	7	8
8	1	4	3	7	5	2	0	6

nótese que s[p(i)] = i

la matriz de distancias d_{ij} *de p es:*

d_{ij} =

0	2	4	4	1	3	3	1	2
2	0	3	2	3	4	1	1	4
4	3	0	1	3	1	2	4	2
4	2	1	0	4	2	1	3	3
1	3	3	4	0	2	4	2	1
3	4	1	2	2	0	3	4	1
3	1	2	1	4	3	0	2	4
1	1	4	3	2	4	2	0	3
2	4	2	3	1	1	4	3	0

como se puede ver $d_{ii} = 0$ y $d_{ij} = d_{ji}$, recuérdese que d_{ij} es la distancia mínima o el número mínimo de slots entre el archivo Fi y el Fj.

la matriz de distancias directas dd_{ij} de p es:

dd_{ij} =

0	2	5	4	8	6	3	1	7
7	0	3	2	6	4	1	8	5
4	6	0	8	3	1	7	5	2
5	7	1	0	4	2	8	6	3
1	3	6	5	0	7	4	2	8
3	5	8	7	2	0	6	4	1
6	8	2	1	5	3	0	7	4
8	1	4	3	7	5	2	0	6
2	4	7	6	1	8	5	3	0

como se puede ver $dd_{ii} = 0$ y $dd_{ij} + dd_{ji} = n$, recuérdese que dd_{ij} es la distancia directa o el número de slots entre el broadcast del archivo Fi y el siguiente broadcast de Fj.

Los valores de la tabla de comparación 4.1.3-a son valores promedio de 7 experimentos realizados, es decir, se realizaron 7 simulaciones y se obtuvieron sus promedios para tener mejores resultados.

Nota: Todos los valores de tiempo obtenidos en este trabajo, tanto los de acceso como los de procesamiento, están en segundos.

Tabla 4.1.3-a Comparación entre el algoritmo de hill climbing y el algoritmo aleatorio para diferentes valores de n, t y files.

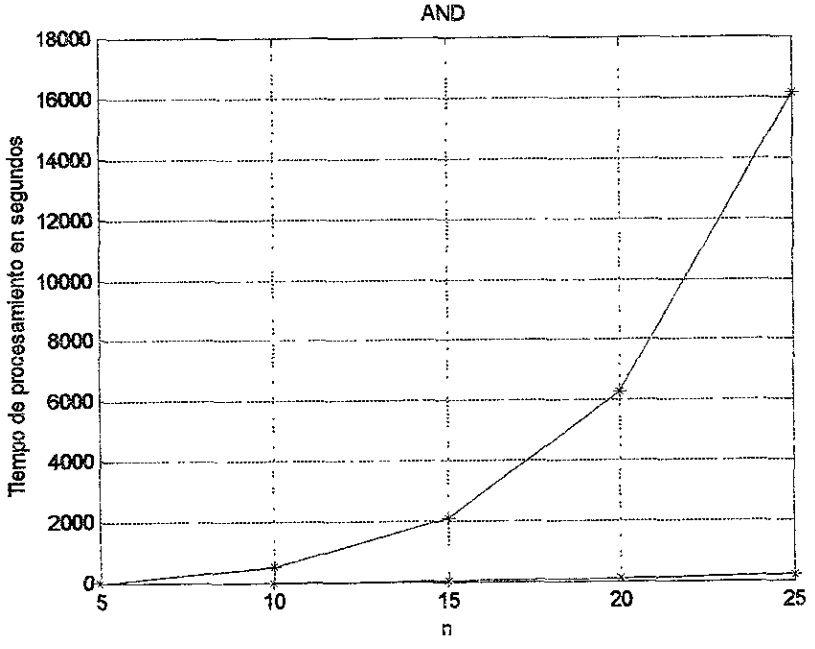
Caso	*t*	*N*	Arreglo "files"	N° total de permutaciones evaluadas con ambos algoritmos	*Tiempo de acceso de H.C.*	*Tiempo de acceso mediante el algoritmo aleatorio*	*Tiempo de procesamiento en segundos con H.C.*	*Tiempo de procesamiento en segundos con el algoritmo aleatorio*
AND	15	5	[2,3,5]	535	3.8002	3.8002	1.9933	21.4033
AND	30	10	[3,5,8,10]	2556	6.3331	6.3370	17.7000	528.1667
AND	45	15	[2,7,10,12,15]	5343	9.2579	9.4564	46.1567	2110.9333
AND	60	20	[4,9,12,15,20]	9314	11.8009	12.3500	86.0333	6304.4667
AND	75	25	[5,8,13,20,25]	14652	14.6831	15.6395	187.2400	16137
OR	15	5	[2,3,5]	522	2.9362	2.9362	1.8133	20.3
OR	30	10	[3,5,8,10]	2448	5.0986	5.1019	12.23	373.1
OR	45	15	[2,7,10,12,15]	5293	6.6188	6.6563	38.0267	1827.5
OR	60	20	[4,9,12,15,20]	8196	8.4513	8.5464	80.6301	5561.4
OR	75	25	[5,8,13,20,25]	11651	9.7356	9.9219	146.6499	11727.3
IMPLY	15	5	[2,3,5]	603	4.1181	4.1181	1.8100	12.5
IMPLY	30	10	[3,5,8,10]	2248	8.0140	8.1899	10.4167	178.9
IMPLY	45	15	[2,7,10,12,15]	5273	12.4061	12.6271	39.2033	1126.6
IMPLY	60	20	[4,9,12,15,20]	8661	16.6485	17.5803	69.4299	2722.5
IMPLY	75	25	[5,8,13,20,25]	11745	21.507	22.6458	118.8099	6377.1

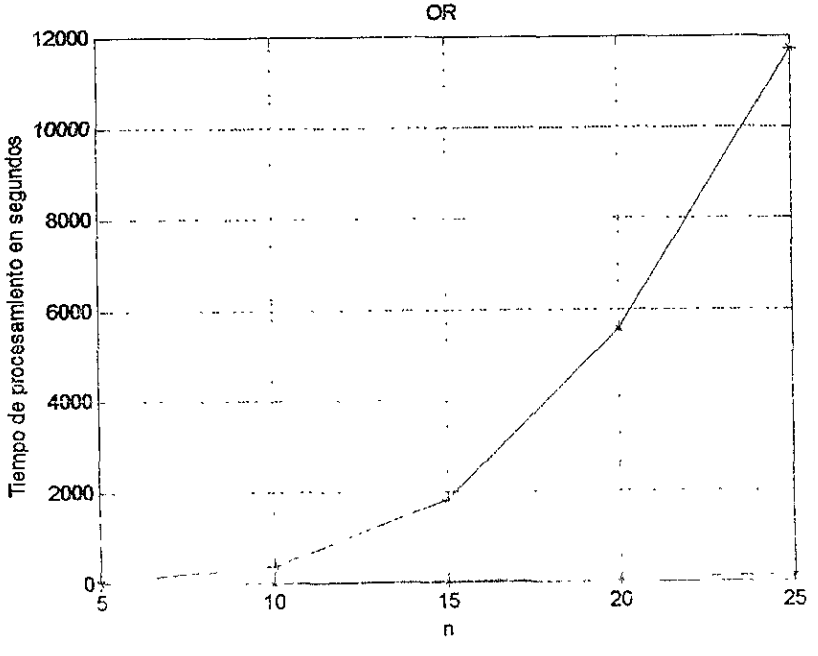
Si se observa la tabla 4.1.3-a, se concluye claramente que el tiempo de acceso que los dos algoritmos arrojan son casi iguales o el de hill climbing es menor por algunas décimas o centécimas de segundo, en donde realmente existe una gran diferencia es en el tiempo de procesamiento, donde el algoritmo de hill climbing es muy superior al algoritmo aleatorio.

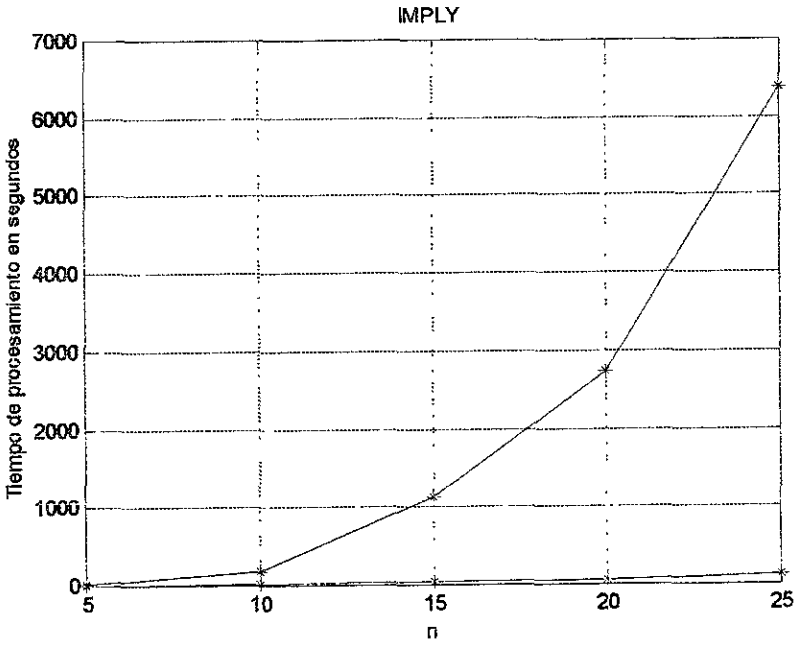
El tiempo de procesamiento del algoritmo de hill climbing crece de forma casi lineal, mientras que el algoritmo aleatorio crece de forma casi cuadrática.

Para generar los resultados de esta tabla se usaron las funciones Final_And, Final_OR y Final_Imply, las cuales se encuentran en el Apéndice B.

A continuación se muestran las gráficas comparativas del tiempo de procesamiento contra el tamaño n de las permutaciones evaluadas


AND
Tiempo de procesamiento en segundos
18000
16000
14000
12000
10000
8000
6000
4000
2000
0
5
10
15
20
25
n

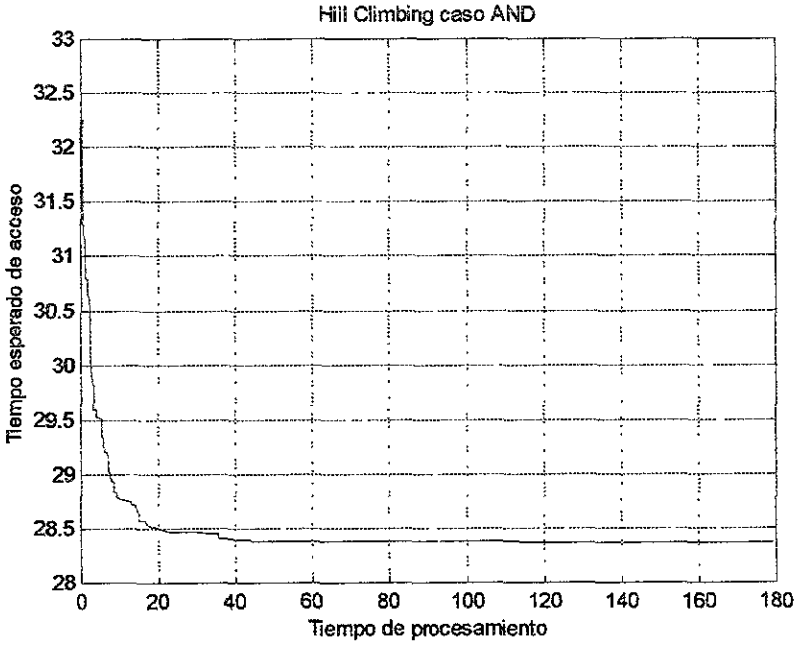

OR
Tiempo de procesamiento en segundos
12000
10000
8000
6000
4000
2000
0
5
10
15
20
25
n

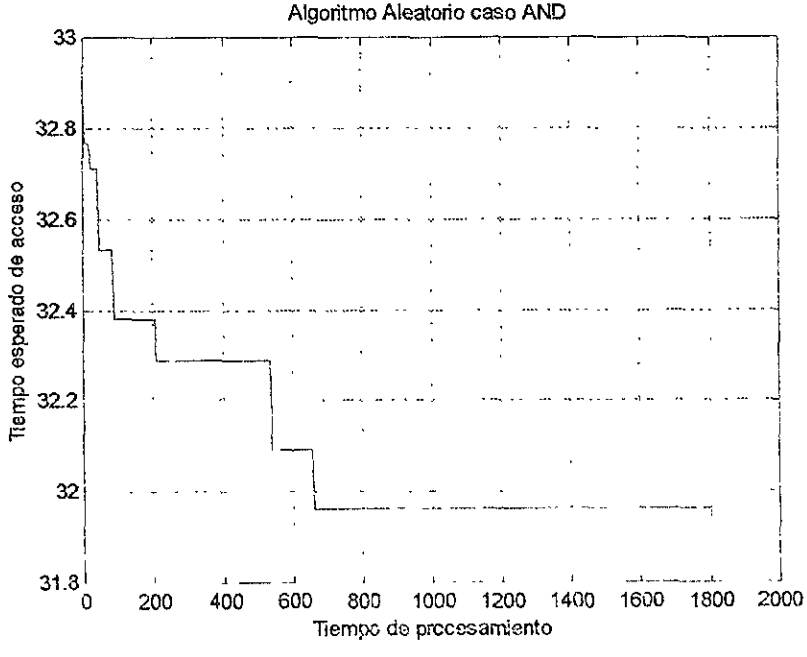


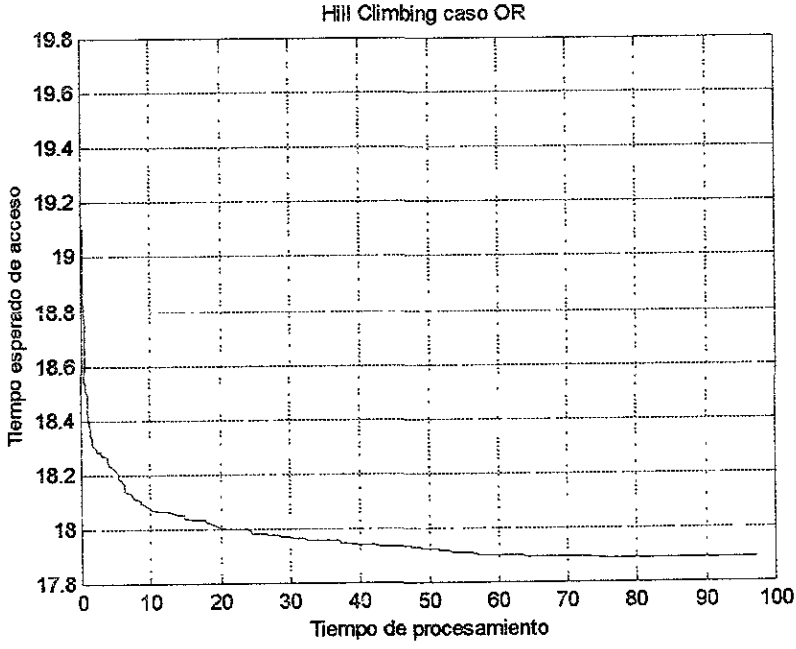
De las tres gráficas se concluye que conforme n aumenta, la diferencia del tiempo de procesamiento se hace más grande, por lo que conforme n aumente convendrá utilizar más el algoritmo de hill climbing que el algoritmo aleatorio.

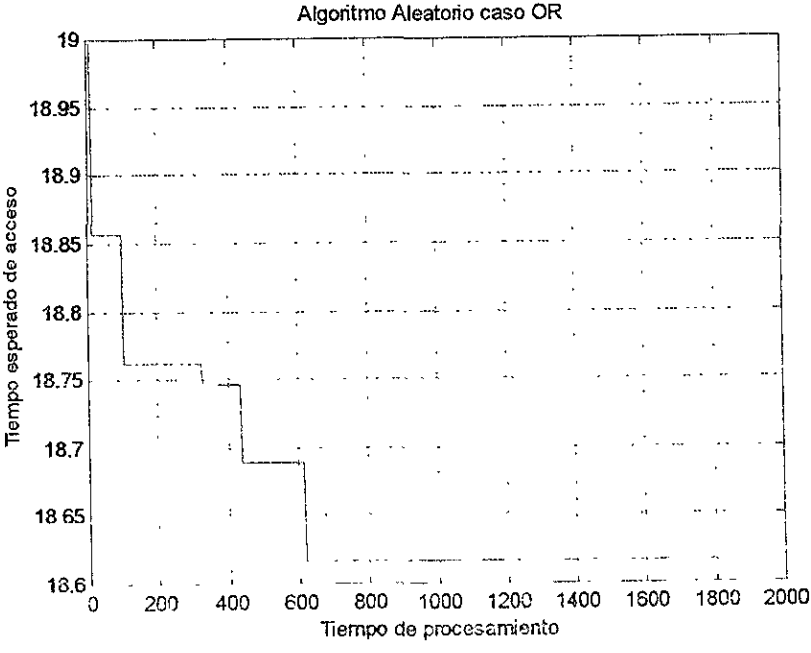
4.1.4 Segunda simulación: Comparación del tiempo de acceso entregado por el algoritmo de hill climbing y el algoritmo aleatorio con tiempos semejantes de procesamiento.

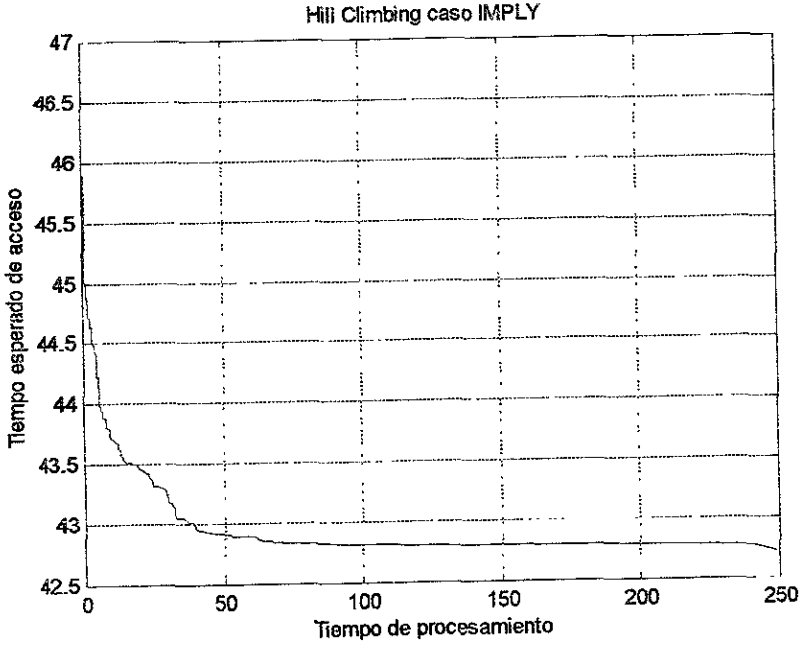
En este caso se programaron las funciones Prog1 la cual recibe dos parámetros, el primero es el tiempo de procesamiento que se quiere usar para evaluar el algoritmo de hill climbing y el algoritmo aleatorio, el segundo es el número t de permutaciones iniciales para evaluar en el algoritmo de hill climbing, en este caso no se hará una tabla comparativa, pienso que son mucho más ilustrativas algunas gráficas. Para esta segunda simulación se generaron 10 permutaciones de 50 archivos, es decir t = 10 permutaciones de tamaño n = 50 con un tiempo máximo de procesamiento de 1800 segundos y el arreglo files = [7,18,22,35,40,42,47,50].
Para la generación de estas gráficas solo se tomaron en cuenta las permutaciones que mejoraban el tiempo de acceso. A continuación se muestran las gráficas del promedio de las diez permutaciones iniciales para el algoritmo de hill climbing y también una gráfica del algoritmo aleatorio para cada caso.

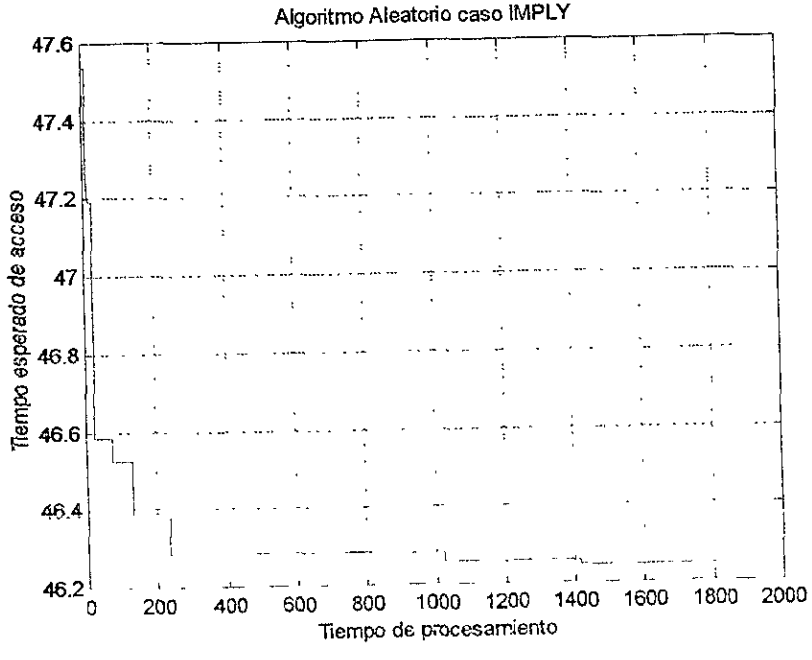

Hill Climbing caso AND
Tiempo esperado de acceso
33
32.5
32
31.5
31
30.5
30
29.5
29
28.5
28
0
20
40
60
80
100
120
140
160
180
Tiempo de procesamiento


Algoritmo Aleatorio caso AND
Tiempo esperado de acceso
33
32.8
32.6
32.4
32.2
32
31.8
0
200
400
600
800
1000
1200
1400
1600
1800
2000
Tiempo de procesamiento


Hill Climbing caso OR
Tiempo esperado de acceso
Tiempo de procesamiento
19.8
19.6
19.4
19.2
19
18.8
18.6
18.4
18.2
18
17.8
0
10
20
30
40
50
60
70
80
90
100


Algoritmo Aleatorio caso OR
Tiempo esperado de acceso
Tiempo de procesamiento
19
18.95
18.9
18.85
18.8
18.75
18.7
18.65
18.6
0
200
400
600
800
1000
1200
1400
1600
1800
2000


Hill Climbing caso IMPLY
Tiempo esperado de acceso
47
46.5
46
45.5
45
44.5
44
43.5
43
42.5
0
50
100
150
200
250
Tiempo de procesamiento


Algoritmo Aleatorio caso IMPLY
Tiempo esperado de acceso
47.6
47.4
47.2
47
46.8
46.6
46.4
46.2
0
200
400
600
800
1000
1200
1400
1600
1800
2000
Tiempo de procesamiento

Observando las seis gráficas anteriores queda claro que para tiempos semejantes de procesamiento computacional el algoritmo de hill climbing entrega mucho mejor tiempo de acceso que el algoritmo aleatorio, también se puede observar que aunque el algoritmo de hill climbing siga trabajando llega un momento en el cual no mejora sus resultados, es decir, se queda en un valor mínimo, mientras que el algoritmo aleatorio conforme pasa el tiempo sigue teniendo alguna mejora, aunque su desventaja es que para llegar a la solución óptima del problema tarda mucho tiempo en procesar.

4.1.5 Tercer simulación: Comparación del tiempo de acceso y el tiempo de procesamiento entregado por el algoritmo de hill climbing y el algoritmo aleatorio con un número dado de permutaciones a evaluar.

Los siguientes resultados se simularon con la función Variando_m(n, maxm, files), la cual su código está en el apéndice B, en donde recibe el tamaño n de las permutaciones a generar, el número total de permutaciones maxm a evaluar y el arreglo files, la tabla muestra el comportamiento de ambos algoritmos conforme se van evaluando las permutaciones, es decir, a diferencia de la pasada simulación en donde solo se graficaron las permutaciones que mejoraban el tiempo de acceso, en esta se graficaron todas para ver el comportamiento en general de ambos algoritmos.

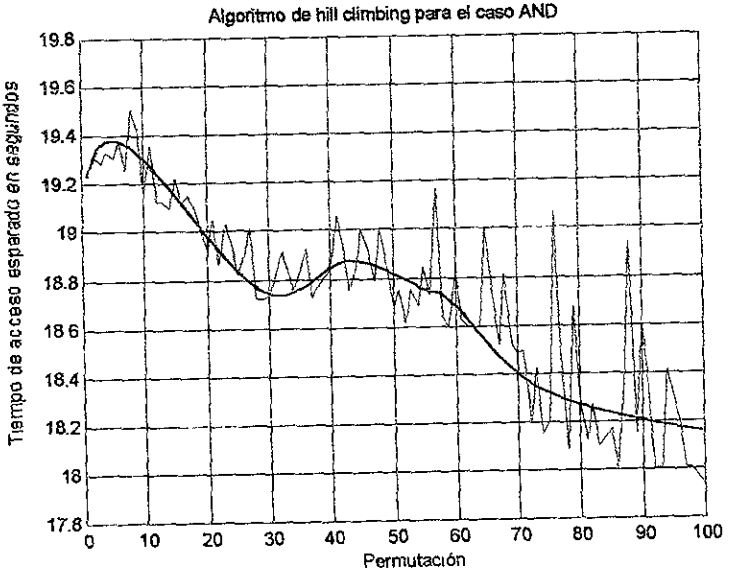
Tabla 4.1.5-a Comparación entre el algoritmo de hill climbing y el algoritmo aleatorio variando el número total de permutaciones a evaluar y manteniendo constante el tamaño n de las permutaciones, en este caso n = 30 y files = [2 7 10 15 18 20 24 26 30], en este caso se partió de una sola permutación inicial.

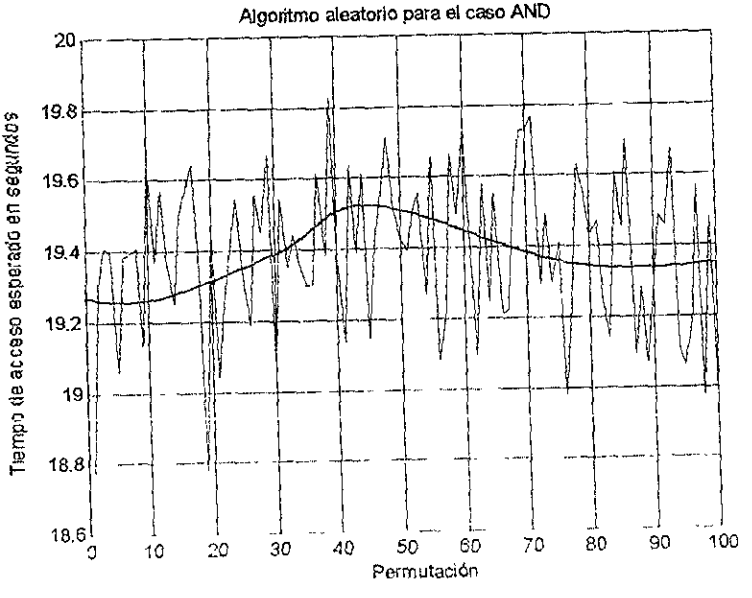
Problema	Nº de permutación evaluada	Tiempo de acceso con H.C.	Tiempo de acceso con el algoritmo aleatorio	Tiempo de procesamiento con H.C.	Tiempo de procesamiento con el algoritmo aleatorio
AND	1	19.2584	18.7717	2.0400	1.7600
AND	10	19.1522	19.6425	2.1500	22.7900
AND	20	18.9267	19.4503	2.2600	52.4000
AND	30	18.7324	19.0610	2.3700	77.7700
AND	40	18.8370	19.3958	2.5300	101.5000
AND	50	18.6846	19.4252	2.5800	125.1200
AND	60	18.8278	19.7333	2.6900	149.1800
AND	70	18.4882	19.7308	2.8000	173.6700
AND	80	18.2890	19.4295	2.9600	197.3500
AND	90	18.6088	19.2994	3.0700	221.6800
AND	100	17.9332	18.9751	3.2400	245.7900
OR	1	12.4379	12.6252	2.6300	2.4200
OR	10	12.2202	12.1847	2.8500	34.5500
OR	20	12.3375	12.3682	3.0700	76.8900
OR	30	12.0763	12.6406	3.2400	103.2000
OR	40	12.4011	12.6225	3.4600	128.2000
OR	50	12.3512	12.3591	3.6800	151.2100
OR	60	12.3178	12.2652	3.9000	174.2300
OR	70	12.3276	12.5527	4.0600	197.4600
OR	80	12.0282	12.4314	4.2800	219.9800
OR	90	12.1528	12.3448	4.5000	242.5500
OR	100	12.0969	12.5072	4.7200	265.4000
IMPLY	1	27.3601	26.7455	1.1600	1.1500
IMPLY	10	27.0111	26.2724	1.3800	11.7000
IMPLY	20	26.2047	26.8411	1.6000	23.3400
IMPLY	30	25.9717	26.8567	1.8200	35.1000
IMPLY	40	25.4420	26.9620	2.0400	46.7400
IMPLY	50	25.2755	27.2387	2.2600	58.6000
IMPLY	60	25.5642	26.6399	2.4800	71.3500
IMPLY	70	25.6960	27.7950	2.7500	84.0400
IMPLY	80	25.0720	27.4602	2.9700	96.6700
IMPLY	90	25.3809	27.0263	3.1900	109.3600

IMPLY	100	25.4739	26.9153	3.4100	122.1000

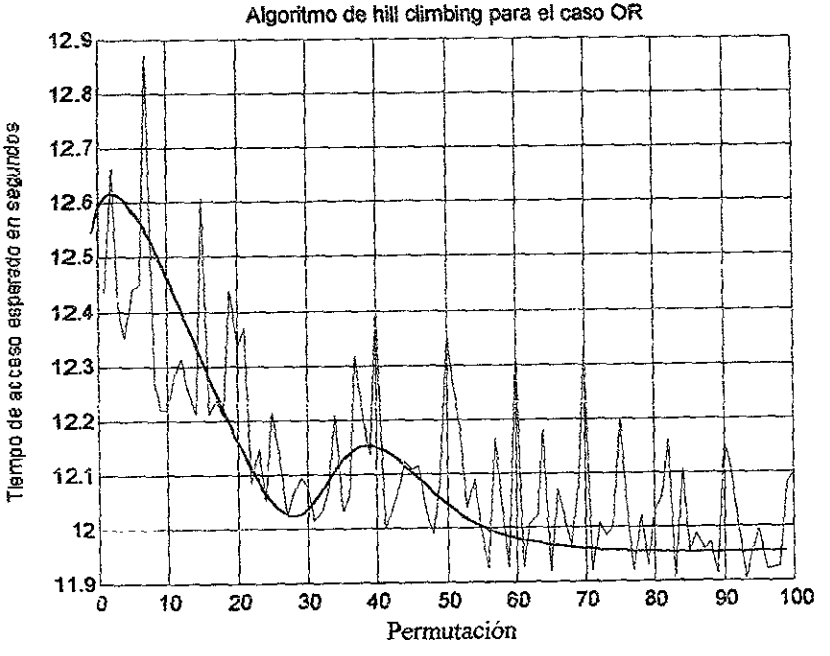
Las gráficas que a continuación se presentan muestran el comportamiento del tiempo de acceso conforme se van evaluando las permutaciones, esto con el fin de comparar el comportamiento de ambos algoritmos conforme aumentan el número de permutaciones a evaluar, la línea azul representa el comportamiento tal como es de ambos algoritmos y la línea roja es un promedio para tener una idea más general.

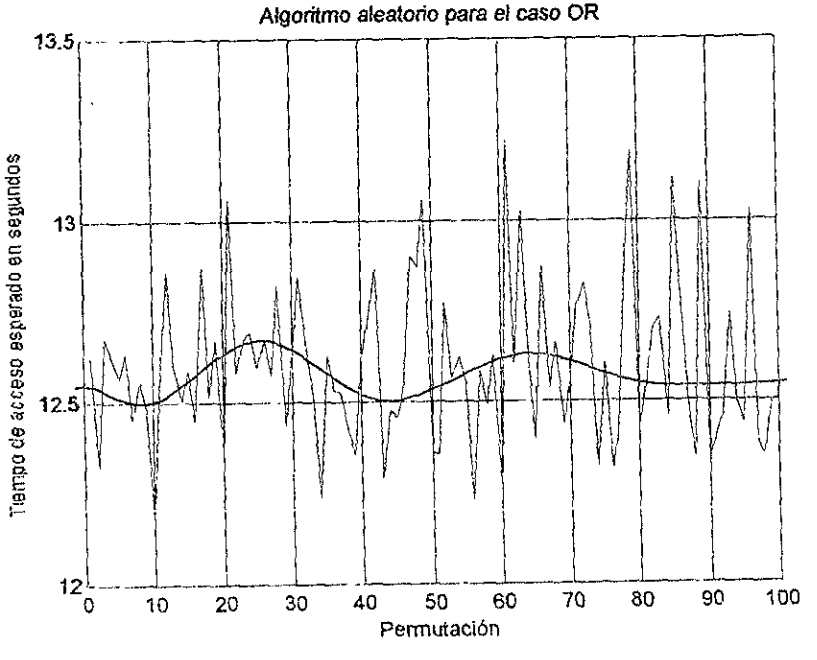
Gráficas 4.1.5.a-1 y 4.1.5.a-2



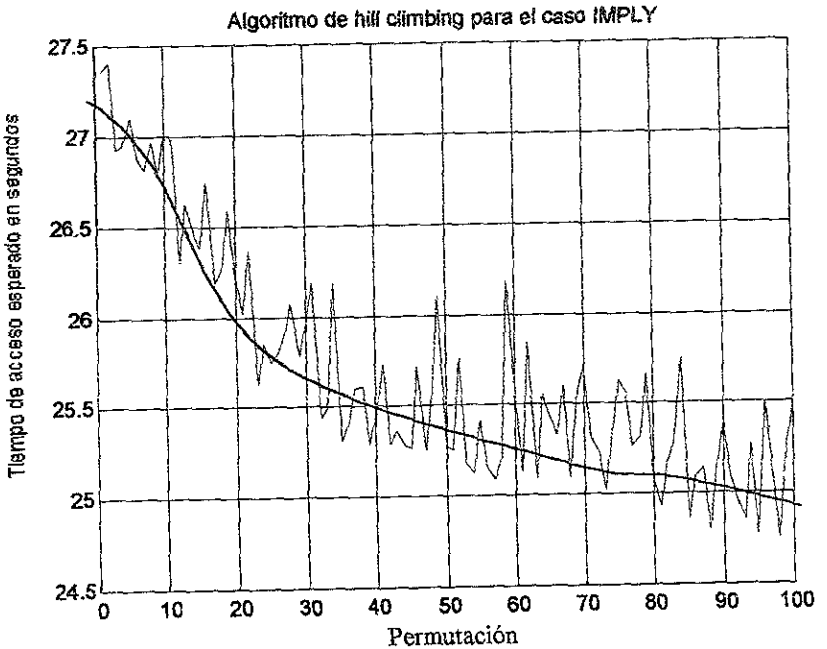


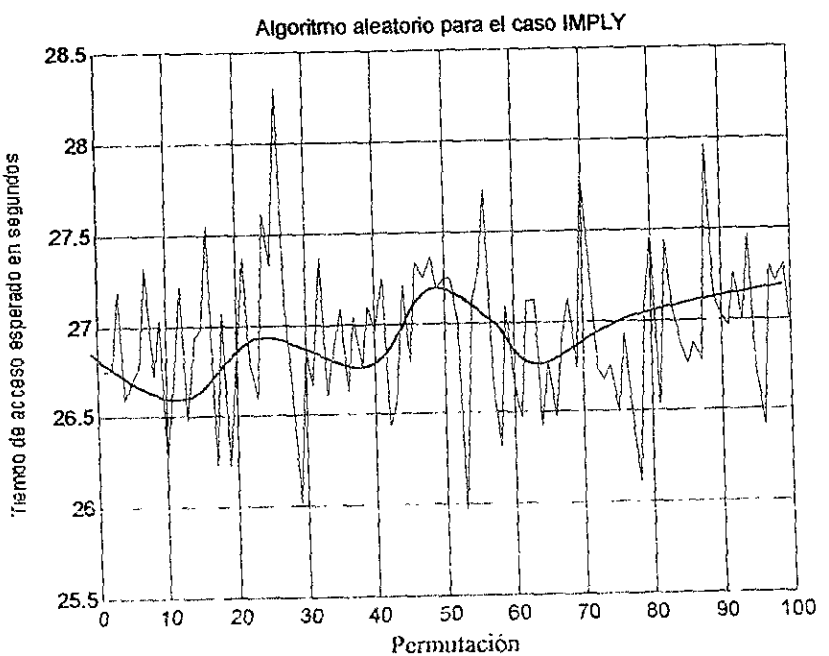
Gráficas 4.1.5.b-1 y 4.1.5.b-2





Gráficas 4.1.5.c-1 y 4.1.5.c-2





Para la tercer simulación se observó que en el algoritmo de hill climbing constantemente se tiende a una bajada, en cambio con el algoritmo aleatorio, como su nombre lo dice se comporta de forma impredecible y sus valores de tiempo esperado se encuentran dentro de un gran rango de valores sin tender a algún valor en especial.

Capítulo V

Contiene las conclusiones y los posibles proyectos que pueden desarrollarse a partir de esta tesis, además de las referencias del material bibliográfico utilizado.

5.1 Conclusiones y futuros trabajos

Recordando los resultados obtenidos en el capítulo 4 en las tabla 4.1.3-a, se puede comparar tanto el tiempo de procesamiento y el tiempo de acceso para los tres problemas estudiados (AND, OR e IMPLY), el tiempo de acceso que los dos algoritmos arrojan son casi iguales o el de hill climbing es menor por algunas décimas o centécimas de segundo, en donde realmente existe una gran diferencia es en el tiempo de procesamiento, donde el algoritmo de hill climbing es muy superior al algoritmo aleatorio, también se puede obsevar de la tabla que los dos algoritmos evaluaron un gran número de permutaciones, es por esta razón que el algoritmo aleatorio alcanza resultados muy semejantes en cuanto a los tiempos de acceso que el algoritmo de hill climbing, en cambio si se revisan las gráficas obtenidas en la segunda y tercera simulación es evidente la forma en como el algoritmo de hill climbing converge a un valor mucho más rápido que el algoritmo aleatorio, por ejemplo en las gráficas de la segunda simulación, si se toman los mismos tiempos de procesamiento el tiempo de acceso de hill climbing es aproximadamente un 10% mejor que el resultado del algoritmo aleaorio, a continuación muestro una tabla comparando los tiempos de acceso cuando se tienen 10 segundos de tiempo de procesamiento y 50 segundos, aunque la mejor comparación se observa en las gráficas ya mencionadas.

Caso	Tiempo de procesamiento	Tiempo de acceso con H.C.	Tiempo de acceso con el algoritmo aleatorio
AND	10 segundos	28.7	32.76
AND	50 segundos	28.4	32.5
OR	10 segundos	18.1	18.86
OR	50 segundos	17.9	18.86
IMPLY	10 segundos	44	47.2
IMPLY	50 segundos	42.9	46.6

La conclusión es que el algoritmo de hill climbing converge a resultados satisfactorios, ya que viendo las gráficas del capítulo 4 para los tres problemas usando el algoritmo de hill climbing se observa que se empieza con una permutación con un tiempo de acceso inicial y conforme se avanza en el número de permutaciones evaluadas el tiempo de acceso va siendo menor, es decir, al aplicar el operador evolutivo de intercambio a las permutaciones, estas se manipulan de tal forma que su tiempo de acceso va siendo menor conforme evolucionan. Los resultados de las gráficas mencionadas contrastan con las obtenidas mediante el algoritmo aleatorio, en donde se observa que puesto que se generan permutaciones de forma aleatoria, sus tiempos de acceso en realidad también son aleatorios, es decir, no tienden a algún valor aunque se aumenten el número de permutaciones evaluadas. En sí el éxito del algoritmo aleatorio se basa en generar un gran número de permutaciones y escoger la que tenga menor tiempo de acceso, mientras que el algoritmo de hill climbing comienza con una población de permutaciones aleatorias y se procesan como ya se explicó de tal forma que se logra obtener el mínimo tiempo de acceso.

También se puede concluir que para un número de permutaciones a evaluar el algoritmo de hill climbing siempre obtiene una permutación con un tiempo de acceso menor que la obtenida por medio del algoritmo aleatorio.

Como se explicó en el capítulo 4 para comparar los resultados de ambos algoritmos, primero se usó el algoritmo de hill climbing contando el número de permutaciones usadas y después se usó ese número de permutaciones evaluadas en el algoritmo aleatorio, esto para comparar lo más fielmente posible los dos algoritmos.

Se concluye que con ambos se puede obtener resultados muy satisfactorios aunque la diferencia aparece en el tiempo de procesamiento computacional, en donde el tiempo tomado para procesar el algoritmo de hill climbing es mucho menor que el del algoritmo aleatorio, como ejemplo los resultados mostrados en la tabla 4.1.3-a muestra que el tiempo máximo que tardó en procesar el algoritmo de hill climbing fue de 2.31 minutos mientras que el tiempo máximo que se obtuvo con el algoritmo aleatorio fue de 3.26 horas, el cual es demasiado tiempo.

Debo aclarar que esta comparación tan exhaustiva del algoritmo de hill climbing con el algoritmo aleatorio se debió a que no se cuenta con otra forma de saber si los resultados obtenidos con el algoritmo de hill climbing son satisfactorios o por lo menos válidos.

En este momento recordaré los objetivos de este trabajo, para concluir si se cumplieron satisfactoriamente:

- Aplicación del algoritmo de Hill Climbing en sistemas asimétricos de telecomunicaciones tipo Clientes - Proveedores - Servidores.

Dado que el algoritmo aleatorio es una técnica que se puede aplicar a este tipo de sistemas y dado que el algoritmo de hill climbing superó al algoritmo aleatorio, se concluye viable su aplicación a los sistemas asimétricos de telecomunicaciones tipo Clientes – Proveedores – Servidores, teniendo en cuenta que cada sistema presenta diferentes características y que en la vida real la interdependencia de archivos puede ser mayor a dos, es decir, se podrían tomar probabilidades entre tres, cuatro o más archivos, creo que en un futuro se podría seguir con este trabajo tomando en cuenta estos puntos.

- Mejoramiento del tiempo de acceso de los usuarios en los sistemas asimétricos de telecomunicaciones tipo Clientes - Proveedores - Servidores.

Este objetivo también se cumplió, ya que se comprobó que mediante el algoritmo de hill climbing se pueden ordenar los archivos en una permutación de tal forma que su tiempo de acceso disminuya. Desgraciadamente si se hace una comparación con el algoritmo aleatorio sin tomar en cuenta el tiempo de procesamiento solo se mejoró el tiempo de acceso por unas cuantas décimas o centécimas de segundo, en este caso agregaría que aunque se tengan los mismos resultados, en el caso del algoritmo aleatorio no hay forma de saber cuantas permutaciones generar para obtener un buen resultado, en cambio con hill climbing el mismo algoritmo determina cuando se ha alcanzado un buen resultado.

- Comparación del tiempo de procesamiento del algoritmo de Hill Climbing y el tiempo de procesamiento obtenido mediante la generación de un número aleatorio de cartas de programación.

En cuanto a este objetivo ya he sido muy enfático a la gran diferencia existente entre el tiempo de procesamiento del algoritmo de hill climbing y el tiempo de procesamiento del algoritmo aleatorio.

Después de ver los resultados obtenidos por el algoritmo de hill climbing debe destacarse el éxito obtenido usando esta técnica heurística, aunque debe recordarse que esta técnica solo garantiza la obtención de un mínimo o un máximo local y a pesar de que se modificó de tal forma que se econtró varios mínimos locales para después elegir el mínimo de todos ellos, esto no garantizó que fuese el mínimo global del problema, además los resultados se obtuvieron suponiendo que las probabilidades se daban solo entre pares de archivos y en la realidad pueden existir probabilidades entre 3, 4 o más archivos, pienso que la presente tesis deja la base para futuras investigaciones de mejorar los sistemas inalámbricos asimétricos tipo Servidores - Proveedores - Clientes. También debe recordarse que esta técnica se aplicó a un modelo anteriormente propuesto, por lo que sería muy bueno que en un futuro se modelen sistemas específicos y se analicen con técnicas heurísticas más avanzadas como los algoritmos genéticos.

5.2 Referencias

- "Pushing Dependent Data in Clients–Providers-Servers Systems"
  Amotz Bar-Noy
  Joseph Seffi Naor
  Baruch Schieber

- "Data Broadcast"
  Jianliang Xu
  Dik-Lun Lee
  Qinglong Hu
  Wang-Chien Lee

- Pagenumber Problem
  Nidhi Kapoor
  Thesis presented to the University of Ottawa
  pp. 38-44

Apéndice A: Resultados y Gráficas adicionales

Caso	*t*	*n*	Arreglo "files"	N° total de permutaciones evaluadas con ambos algoritmos	*Tiempo de acceso de H.C.*	*Tiempo de acceso mediante el algoritmo aleatorio*	*Tiempo de procesamiento en segundos con H.C.*	*Tiempo de procesamiento en segundos con el algoritmo aleatorio*
AND	15	5	[2,3,5]	535	3.8002	3.8002	1.9933	21.4033
AND	30	10	[3,5,8,10]	2556	6.3331	6.3370	17.7000	528.1667
AND	45	15	[2,7,10,12,15]	5343	9.2579	9.4564	46.1567	2110.9333
AND	60	20	[4,9,12,15,20]	9314	11.8009	12.3500	86.0333	6304.4667
AND	75	25	[5,8,13,20,25]	14652	14.6831	15.6395	187.2400	16137
OR	15	5	[2,3,5]	522	2.9362	2.9362	1.8133	20.3
OR	30	10	[3,5,8,10]	2448	5.0986	5.1019	12.23	373.1
OR	45	15	[2,7,10,12,15]	5293	6.6188	6.6563	38.0267	1827.5
OR	60	20	[4,9,12,15,20]	8196	8.4513	8.5464	80.6301	5561.4
OR	75	25	[5,8,13,20,25]	11651	9.7356	9.9219	146.6499	11727.3
IMPLY	15	5	[2,3,5]	603	4.1181	4.1181	1.8100	12.5
IMPLY	30	10	[3,5,8,10]	2248	8.0140	8.1899	10.4167	178.9
IMPLY	45	15	[2,7,10,12,15]	5273	12.4061	12.6271	39.2033	1126.6
IMPLY	60	20	[4,9,12,15,20]	8661	16.6485	17.5803	69.4299	2722.5
IMPLY	75	25	[5,8,13,20,25]	11745	21.507	22.6458	118.8099	6377.1

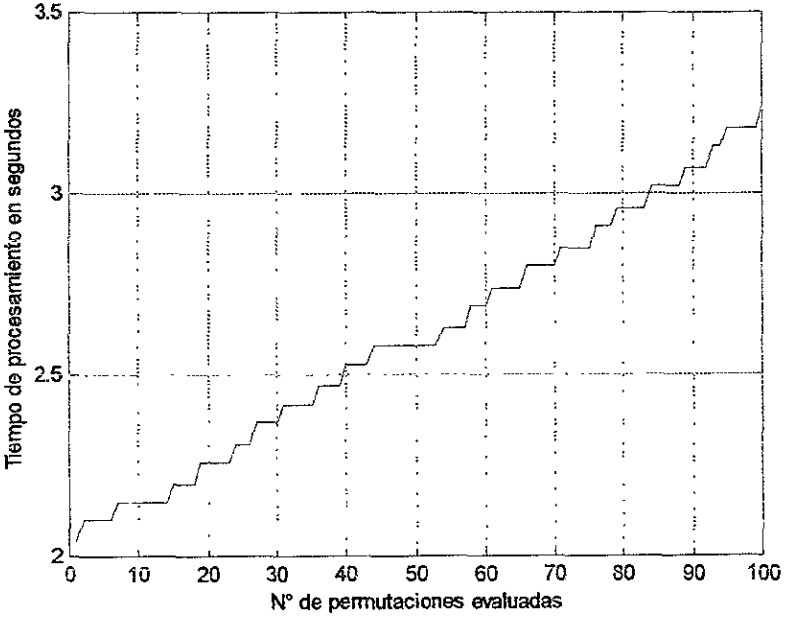
Caso	*t*	*n*	Arreglo "files"	Permutación con H.C.	Permutación con alg. aleatorio
AND	15	5	[2,3,5]	0 2 4 3 1	2 0 1 3 4
AND	30	10	[3,5,8,10]	0 4 3 8 9 5 7 6 1 2	7 5 9 8 3 4 0 2 1 6
AND	45	15	[2,7,10,12,15]	6 5 8 9 7 10 11 12 13 14 0 1 2 3 4	1 2 6 5 4 3 12 11 10 14 13 9 7 0 8
AND	60	20	[4,9,12,15,20]	10 11 9 4 5 7 8 6 2 3 1 0 18 16 17 15 19 13 12 14	13 11 2 3 0 1 14 8 6 5 12 17 4 16 18 15 19 9 7 10
AND	75	25	[5,8,13,20,25]	10 11 17 19 16 13 14 18 15 3 2 4 0 1 7 6 5 21 24 20 23 22 12 9 8	12 0 1 17 18 19 14 23 15 13 20 24 22 16 6 7 21 10 2 3 8 11 9 5 4
OR	15	5	[2,3,5]	0 3 1 4 2	0 2 4 1 3
OR	30	10	[3,5,8,10]	9 0 4 1 7 8 3 2 5 6	8 4 2 5 6 9 0 3 7 1
OR	45	15	[2,7,10,12,15]	10 7 12 4 2 1 8 14 11 3 6 13 9 0 5	4 0 14 11 6 3 7 12 9 1 10 5 13 2 8
OR	60	20	[4,9,12,15,20]	18 5 3 19 4 1 9 15 8 13 16 10 0 6 2 7 17 12 11 14	18 7 6 19 11 15 3 2 14 16 5 1 17 4 9 17 8 0 10 13
OR	75	25	[5,8,13,20,25]	22 3 17 15 20 5 0 19 21 8 17 6 1 14 2 9 24 13 23 11 16 4 7 10 18	7 8 17 4 19 9 24 22 6 3 16 11 10 13 15 20 21 5 0 12 18 1 14 23
IMPLY	15	5	[2,3,5]	3 0 1 2 4	4 3 0 1 2

IMPLY	30	10	[3,5,8,10]	0 5 7 6 4 3 9 8 2 1	2 6 1 0 9 8 4 5 7 3
IMPLY	45	15	[2,7,10,12,15]	12 14 9 7 3 2 6 4 5 1 0 10 11 8 13	4 11 1 5 0 9 13 12 14 7 3 2 10 6 8
IMPLY	60	20	[4,9,12,15,20]	14 2 5 4 7 8 13 0 6 1 3 10 16 17 18 19 15 9 11 12	13 16 12 2 5 18 0 19 7 15 4 1 3 14 9 8 6 10 11 17
IMPLY	75	25	[5,8,13,20,25]	3 22 14 17 13 6 8 9 11 7 5 10 20 24 15 12 23 21 4 1 0 2 19 16 18	3 20 19 17 13 11 12 16 18 23 24 22 4 8 6 9 21 14 1 0 2 10 5 7 15

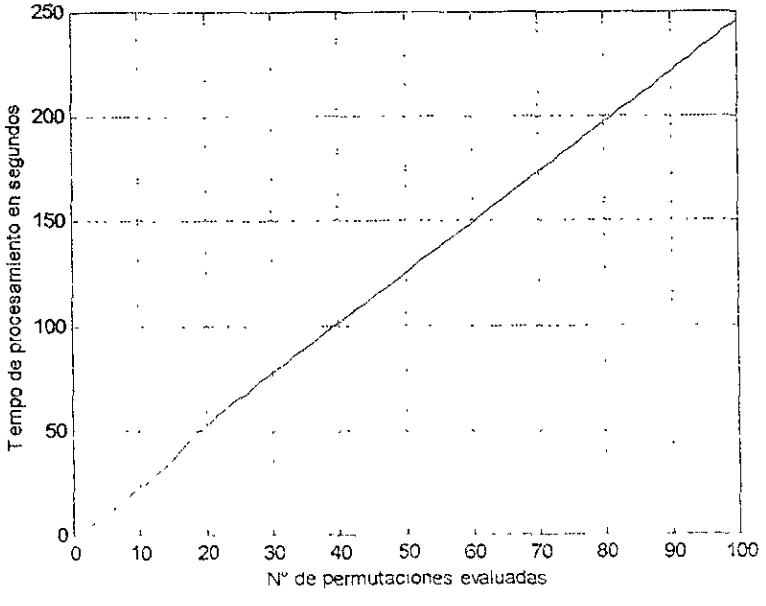
n=30 y files = [2 7 10 15 18 20 24 26 30]

Problema	N° de permutaciones evaluadas	Tiempo de acceso con H.C.	Tiempo de acceso con el algoritmo aleatorio	Tiempo de procesamiento con H.C.	Tiempo de procesamiento con el algoritmo aleatorio
AND	1	19.2584	18.7717	2.0400	1.7600
AND	10	19.1522	19.6425	2.1500	22.7900
AND	20	18.9267	19.4503	2.2600	52.4000
AND	30	18.7324	19.0610	2.3700	77.7700
AND	40	18.8370	19.3958	2.5300	101.5000
AND	50	18.6846	19.4252	2.5800	125.1200
AND	60	18.8278	19.7333	2.6900	149.1800
AND	70	18.4882	19.7308	2.8000	173.6700
AND	80	18.2890	19.4295	2.9600	197.3500
AND	90	18.6088	19.2994	3.0700	221.6800
AND	100	17.9332	18.9751	3.2400	245.7900
OR	1	12.4379	12.6252	2.6300	2.4200
OR	10	12.2202	12.1847	2.8500	34.5500
OR	20	12.3375	12.3682	3.0700	76.8900
OR	30	12.0763	12.6406	3.2400	103.2000
OR	40	12.4011	12.6225	3.4600	128.2000
OR	50	12.3512	12.3591	3.6800	151.2100
OR	60	12.3178	12.2652	3.9000	174.2300
OR	70	12.3276	12.5527	4.0600	197.4600
OR	80	12.0282	12.4314	4.2800	219.9800
OR	90	12.1528	12.3448	4.5000	242.5500
OR	100	12.0969	12.5072	4.7200	265.4000
IMPLY	1	27.3601	26.7455	1.1600	1.1500
IMPLY	10	27.0111	26.2724	1.3800	11.7000
IMPLY	20	26.2047	26.8411	1.6000	23.3400
IMPLY	30	25.9717	26.8567	1.8200	35.1000
IMPLY	40	25.4420	26.9620	2.0400	46.7400
IMPLY	50	25.2755	27.2387	2.2600	58.6000
IMPLY	60	25.5642	26.6399	2.4800	71.3500
IMPLY	70	25.6960	27.7950	2.7500	84.0400
IMPLY	80	25.0720	27.4602	2.9700	96.6700
IMPLY	90	25.3899	27.0263	3.1900	109.3600
IMPLY	100	25.4739	26.9153	3.4100	122.1000

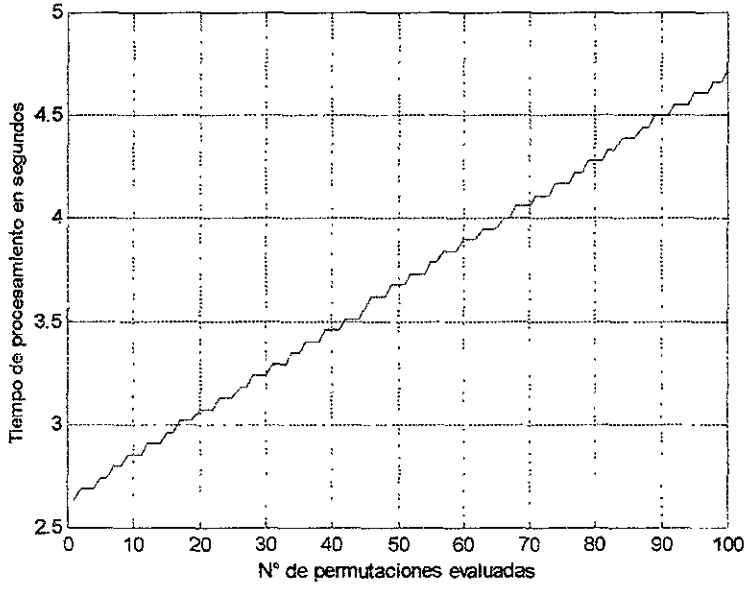
"Hill climbing caso AND"



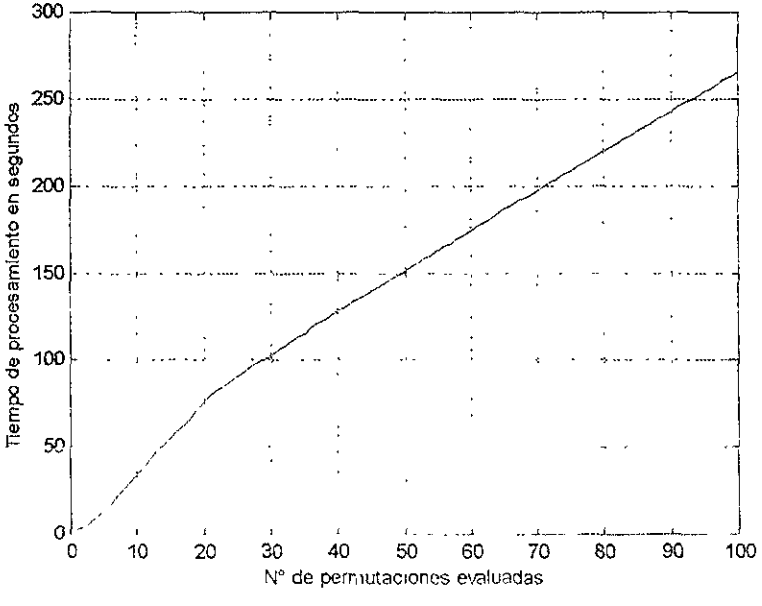
"Algoritmo aleatorio caso AND"



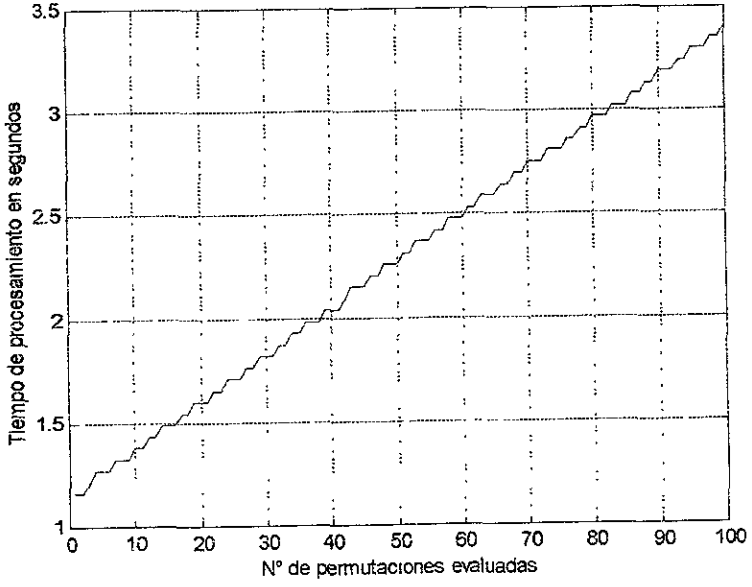
"Hill climbing caso OR"



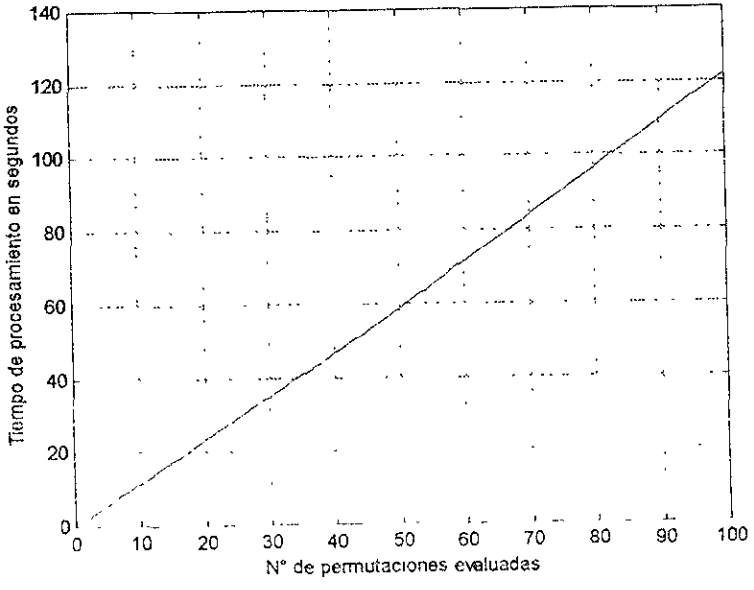
"Algoritmo aleatorio caso OR"



"Hill climbing caso IMPLY"



"Algoritmo aleatorio caso IMPLY"



Apéndice B: Código

```
/*-----------------------------------------------------------------------*/

function Final_And

n = [5 10 15 20 25];
t = 3*n;
files1 = [2 3 5];
files2 = [3 5 8 10];
files3 = [2 7 10 12 15];
files4 = [4 9 12 15 20];
files5 = [5 8 13 20 25];

Hill_Xpi_prom = zeros(7,1);
Hill_Comput_prom = zeros(7,1);
Random_Xpi_prom = zeros(7,1);
Random_Comput_prom = zeros(7,1);

for j = 1:7
   for i = 1:5

      switch i
         case 1
            files = files1;
         case 2
            files = files2;
         case 3
            files = files3;
         case 4
            files = files3;
         case 5
            files = files5;
      end

      probabilts(n(i), files); %generates aij, bij and cij and save it in the
file probs
      load('probs'); % loads the variables aij, bij and cij

      diary probabilities.txt
         bij
      diary off;


   [And_Hill_pi,And_total_count,Hill_Api,And_Hill_time]=And_t(t(i),n(i),aij);
%H.C. algorithm for AND

   [And_Random_pi,Random_Api,And_Random_time]= Rand_And(And_total_count,n(i),a
ij);%R. algorithm for AND

      diary And_final.txt;

      )
      )
```

```
      t(i)
      n(i)
      And_total_count
      And_Hill_pi
      And_Random_pi
      Hill_Api
      Random_Api
      And_Hill_time
      And_Random_time

      diary off;

      Hill_Xpi_prom(j,1)=Hill_Xpi_prom(j,1) + Hill_Api;
      Hill_Comput_prom(j,1)=Hill_Comput_prom(j,1) + And_Hill_time;
      Random_Xpi_prom(j,1)=Random_Xpi_prom(j,1) + Random_Api;
      Random_Comput_prom(j,1)=Random_Comput_prom(j,1) + And_Random_time;

   end
end

Hill_Xpi_prom = Hill_Xpi_prom / 7;
Hill_Comput_prom = Hill_Comput_prom / 7;
Random_Xpi_prom = Random_Xpi_prom / 7;
Random_Comput_prom = Random_Comput_prom / 7;

diary And_promedios.txt

   Hill_Xpi_prom
   Hill_Comput_prom
   Random_Xpi_prom
   Random_Comput_prom

diary off;

/*-------------------------------------------------------------------------*/

function Final_Or

n = [5 10 15 20 25];
t = 3*n;
files1 = [2 3 5];
files2 = [3 5 8 10];
files3 = [2 7 10 12 15];
files4 = [4 9 12 15 20];
files5 = [5 8 13 20 25];

Hill_Xpi_prom = zeros(7,1);
Hill_Comput_prom = zeros(7,1);
Random_Xpi_prom = zeros(7,1);
Random_Comput_prom = zeros(7,1);

for j=1:7
   for i=1:5

      switch i
         case 1
            files = files1;
```

```
         case 2
            files = files2;
         case 3
            files = files3;
         case 4
            files = files3;
         case 5
            files = files5;
      end

      probabilts(n(i), files); %generates aij, bij and cij and save it in the
file probs
      load('probs'); % loads the variables aij, bij and cij

      diary probabilities.txt
         bij
      diary off;

      [Or_Hill_pi,Or_total_count,Hill_Bpi,Or_Hill_time] =
Or_t(t(i),n(i),bij);%H.C. algorithm for OR
      [Or_Random_pi,Random_Bpi,Or_Random_time] =
Rand_Or(Or_total_count,n(i),bij);%R. algorithm for OR

      diary Or_final.txt;

      j
      i
      t(i)
      n(i)
      Or_total_count
      Or_Hill_pi
      Or_Random_pi
      Hill_Bpi
      Random_Bpi
      Or_Hill_time
      Or_Random_time

      diary off;

      Hill_Xpi_prom(j,1) = Hill_Xpi_prom(j,1) + Hill_Bpi;
      Hill_Comput_prom(j,1) = Hill_Comput_prom(j,1) + Or_Hill_time;
      Random_Xpi_prom(j,1) = Random_Xpi_prom(j,1) + Random_Bpi;
      Random_Comput_prom(j,1) = Random_Comput_prom(j,1) + Or_Random_time;

   end
end

Hill_Xpi_prom = Hill_Xpi_prom / 7;
Hill_Comput_prom = Hill_Comput_prom / 7;
Random_Xpi_prom = Random_Xpi_prom / 7;
Random_Comput_prom = Random_Comput_prom / 7;

diary Or_promedios.txt

   Hill_Xpi_prom
   Hill_Comput_prom
   Random_Xpi_prom
```

```
   Random_Comput_prom

diary off;

/*-----------------------------------------------------------------------*/

function Final_Imply

n = [5 10 15 20 25];
t = 3*n;
files1 = [2 3 5];
files2 = [3 5 8 10];
files3 = [2 7 10 12 15];
files4 = [4 9 12 15 20];
files5 = [5 8 13 20 25];

Hill_Xpi_prom = zeros(7,1);
Hill_Comput_prom = zeros(7,1);
Random_Xpi_prom = zeros(7,1);
Random_Comput_prom = zeros(7,1);

for j = 1:7
   for i=1:5

      switch i
         case 1
            files = files1;
         case 2
            files = files2;
         case 3
            files = files3;
         case 4
            files = files3;
         case 5
            files = files5;
      end


   [Imply_Hill_pi,Imply_total_count,Hill_Cpi,Imply_Hill_time]=Imply_t(t(i),n(
i),cij);%H.C. algorithm for IMPLY

   [Imply_Random_pi,Random_Cpi,Imply_Random_time]=Rand_Imply(Imply_total_coun
t,n(i),cij);%R. algorithm for IMPLY

      diary Imply_final.txt;

      i
      t(i)
      n(j)
      Imply_total_count
      Imply_Hill_pi
      Imply_Random_pi
      Hill_Cpi
      Random_Cpi
      Imply_Hill_time
      Imply_Random_time
```

```
      diary off;

      Hill_Xpi_prom(j,1) = Hill_Xpi_prom(j,1) + Hill_Cpi;
      Hill_Comput_prom(j,1) = Hill_Comput_prom(j,1) + Imply_Hill_time;
      Random_Xpi_prom(j,1) = Random_Xpi_prom(j,1) + Random_Cpi;
      Random_Comput_prom(j,1) = Random_Comput_prom(j,1) + Imply_Random_time;
   end
end

Hill_Xpi_prom = Hill_Xpi_prom / 7;
Hill_Comput_prom = Hill_Comput_prom / 7;
Random_Xpi_prom = Random_Xpi_prom / 7;
Random_Comput_prom = Random_Comput_prom / 7;

diary Imply_promedios.txt

   Hill_Xpi_prom
   Hill_Comput_prom
   Random_Xpi_prom
   Random_Comput_prom

diary off;

/*-----------------------------------------------------------------------*/

function Prog1(total_time,t)

load('probs'); % loads the variables aij, bij and cij

diary probabilities.txt
   aij
   bij
   cij
diary off;

diary files.txt
   files = [7 18 22 35 40 42 47 50]
diary off;

n=50;

for i=1:t

[And_Hill_pi,And_Hill_array,And_Hill_time]=Hill_And_time(aij,n,total_time);
   diary Hill_And.txt
      i
      And_Hill_array(:,2)
      And_Hill_array(:,1)
      And_Hill_pi
      And_Hill_time
   diary off;

   [Or_Hill_pi,Or_Hill_array,Or_Hill_time]=Hill_Or_time(bij,n,total_time);
   diary Hill_Or.txt
      i
      Or_Hill_array(:,2)
      Or_Hill_array(:,1)
```

```
      Or_Hill_pi
      Or_Hill_time
   diary off;

   [Imply_Hill_pi,Imply_Hill_array,Imply_Hill_time]=Hill_Imply_time(cij,n,tot
al_time);
   diary Hill_Imply.txt
      i
      Imply_Hill_array(:,2)
      Imply_Hill_array(:,1)
      Imply_Hill_pi
      Imply_Hill_time
   diary off;

   figure;
   plot(And_Hill_array(:,2),And_Hill_array(:,1));
   title('Hill Climbing caso AND');
   xlabel('Tiempo de procesamiento');
   ylabel('Tiempo esperado de acceso');
   grid on;

   figure;
   plot(Or_Hill_array(:,2),Or_Hill_array(:,1));
   title('Hill Climbing caso OR');
   xlabel('Tiempo de procesamiento');
   ylabel('Tiempo esperado de acceso');
   grid on;

   figure;
   plot(Imply_Hill_array(:,2),Imply_Hill_array(:,1));
   title('Hill Climbing caso IMPLY');
   xlabel('Tiempo de procesamiento');
   ylabel('Tiempo esperado de acceso');
   grid on;

end

[And_Random_pi,And_Random_array,And_Random_time]=Rand_And_time(aij,n,total_ti
me);
[Or_Random_pi,Or_Random_array,Or_Random_time]=Rand_Or_time(bij,n,total_time);
[Imply_Random_pi,Imply_Random_array,Imply_Random_time]=Rand_Imply_time(cij,n,
total_time);

diary Random_And.txt
   And_Random_array(:,2)
   And_Random_array(:,1)
   And_Random_pi
   And_Random_time
diary off;

diary Random_Or.txt
   Or_Random_array(:,2)
   Or_Random_array(:,1)
   Or_Random_pi
   Or_Random_time
diary off;
```

```
diary Random_Imply.txt
   Imply_Random_array(:,2)
   Imply_Random_array(:,1)
   Imply_Random_pi
   Imply_Random_time
diary off;

figure;
plot(And_Random_array(:,2),And_Random_array(:,1));
title('Algoritmo Aleatorio caso AND');
xlabel('Tiempo de procesamiento');
ylabel('Tiempo esperado de acceso');
grid on;

figure;
plot(Or_Random_array(:,2),Or_Random_array(:,1));
title('Algoritmo Aleatorio caso OR');
xlabel('Tiempo de procesamiento');
ylabel('Tiempo esperado de acceso');
grid on;

figure;
plot(Imply_Random_array(:,2),Imply_Random_array(:,1));
title('Algoritmo Aleatorio caso IMPLY');
xlabel('Tiempo de procesamiento');
ylabel('Tiempo esperado de acceso');
grid on;

/*-----------------------------------------------------------------*/

function Variando_m(n,maxm, files)

probabilts(n, files); %generates aij, bij and cij and save it in the file
probs
load('probs'); % loads the variables aij, bij and cij

diary probabilities.txt
   aij
   bij
   cij
diary off;

diary files.txt
   files
diary off;

[And_Hill_pi,And_Hill_array]=Hill_And_m(aij,n,maxm);
[And_Rand_pi,And_Rand_array]=Rand_And_m(aij,n,maxm);

[Or_Hill_pi,Or_Hill_array]=Hill_Or_m(bij,n,maxm);
[Or_Rand_pi,Or_Rand_array]=Rand_Or_m(bij,n,maxm);

[Imply_Hill_pi,Imply_Hill_array]=Hill_Imply_m(cij,n,maxm);
[Imply_Rand_pi,Imply_Rand_array]=Rand_Imply_m(cij,n,maxm);

diary Hill_And.txt
   And_Hill_pi
```

```
   And_Hill_array
diary off;

diary Rand_And.txt
   And_Rand_pi
   And_Rand_array
diary off;

diary Hill_Or.txt
   Or_Hill_pi
   Or_Hill_array
diary off;

diary Rand_Or.txt
   Or_Rand_pi
   Or_Rand_array
diary off;

diary Hill_Imply.txt
   Imply_Hill_pi
   Imply_Hill_array
diary off;

diary Rand_Imply.txt
   Imply_Rand_pi
   Imply_Rand_array
diary off;

% Graphs for the AND case
figure;
plot(And_Hill_array(1,:));
title('Algoritmo de hill climbing para el caso AND');
xlabel('N° de permutaciones evaluadas');
ylabel('Tiempo de acceso esperado en segundos');
grid on;
figure;
plot(And_Hill_array(2,:));
xlabel('N° de permutaciones evaluadas');
ylabel('Tiempo de procesamiento en segundos');
grid on;
figure;
plot(And_Rand_array(1,:));
title('Algoritmo aleatorio para el caso AND');
xlabel('N° de permutaciones evaluadas');
ylabel('Tiempo de acceso esperado en segundos');
grid on;
figure;
plot(And_Rand_array(2,:));
xlabel('N° de permutaciones evaluadas');
ylabel('Tiempo de procesamiento en segundos');
grid on;

% Graphs for the OR case
figure;
plot(Or_Hill_array(1,:));
title('Algoritmo de hill climbing para el caso OR');
xlabel('N° de permutaciones evaluadas');
```

```
ylabel('Tiempo de acceso esperado en segundos');
grid on;
figure;
plot(Or_Hill_array(2,:));
xlabel('N° de permutaciones evaluadas');
ylabel('Tiempo de procesamiento en segundos');
grid on;
figure;
plot(Or_Rand_array(1,:));
title('Algoritmo aleatorio para el caso OR');
xlabel('N° de permutaciones evaluadas');
ylabel('Tiempo de acceso esperado en segundos');
grid on;
figure;
plot(Or_Rand_array(2,:));
xlabel('N° de permutaciones evaluadas');
ylabel('Tiempo de procesamiento en segundos');
grid on;

% Graphs for the IMPLY case
figure;
plot(Imply_Hill_array(1,:));
title('Algoritmo de hill climbing para el caso IMPLY');
xlabel('N° de permutaciones evaluadas');
ylabel('Tiempo de acceso esperado en segundos');
grid on;
figure;
plot(Imply_Hill_array(2,:));
xlabel('N° de permutaciones evaluadas');
ylabel('Tiempo de procesamiento en segundos');
grid on;
figure;
plot(Imply_Rand_array(1,:));
title('Algoritmo aleatorio para el caso IMPLY');
xlabel('N° de permutaciones evaluadas');
ylabel('Tiempo de acceso esperado en segundos');
grid on;
figure;
plot(Imply_Rand_array(2,:));
xlabel('N° de permutaciones evaluadas');
ylabel('Tiempo de procesamiento en segundos');
grid on;

/*-----------------------------------------------------------------------*/

function [best_pi,total_count,Api_min,time]=And_t(t,n,aij)
%AND t gives the best pi permutation of t
%permutations resulting from applying the hill
%climbing algorithm to t random permutations
%the permutations are of size n.

tic;
p=genera_pi(n);
s=sigm(p);
d=di(s);
[best_pi,total_count,Api_min]=Hill_And(aij,d,p);
```

```
if t > 1
   for i=2:t
      p=genera_pi(n);
      s=sigm(p);
      d=dij(s);
      [new_pi_temp,tc,Apim_temp]=Hill_And(aij,d,p);
      total_count=total_count+tc;
      if Apim_temp < Api_min
         best_pi=new_pi_temp;
         Api_min=Apim_temp;
      end
   end
end
time=toc;

/*-----------------------------------------------------------------*/

function array=genera_pi(n)
%Generates permutation pi or the infinite cyclic schedule
%n is the size of the schedule or the number of files

array=zeros(2,n);            %array is the resulting schedule
array(1,:)=0:n-1;            %The first row of array is the position
array(2,:)=randperm(n)-1;    %The second row is the random permutation
                             %of the schedule

/*-----------------------------------------------------------------*/

function sig=sigm(pis)
%generates sigma (inverse of function pi)
%pis is the schedule to broadcast

sig=pis;        %sig is the resulting sigma
n=length(pis);

if n>1
   for i=1:n
      j=1;
      while (pis(1,i) ~= pis(2,j))
         %this cycle serves for mapping one slot of time
         %all pi for obtaining sigma
         j=j+1;
      end
      sig(2,i)=pis(1,j);
   end
end

/*-----------------------------------------------------------------*/

function d=dij(sig)
% returns the shorter cyclic distance between all the files
% in the permutation pi (pi is the inverse of sig)
% It's a symmetric function, i.e., dij=dji.
% d is the matrix with all the distances

n=length(sig);
d=zeros(n,n);
```

```
for i=1:n
   for j=1:n
      d(i,j)=distance(i,j,sig);    %explained in its prototype
   end
end

/*-------------------------------------------------------------------*/

function dij=distance(i,j,sig)
%Gives the distance between files Fi and
%Fj in the infinite cyclic schedule pi
%The input is sigma because its more easy to manage

n=length(sig);
dij=[mod(sig(2,i)-sig(2,j),n) mod(sig(2,j)-sig(2,i),n)];
%first dij is a vector 1x2
%where dij(1,1) is the forward distance
%and dij(1,2) is the backward distance

dij=min(dij);
%the distance is the minimum between the forward end backward
%distances

/*-------------------------------------------------------------------*/

function [new_pi,total_count,Api_min]=Hill_And(aij,d,p_i)
% Hill Climbing for finding a local minimum for
% a given permutation in the AND problem.

A_pi=Api(aij,d,p_i);

total_count=0; % number of permutations for which
               % fitness function will be evaluate.

max=0;         % this helps to determine the minimum

while max < 30
   [Anuevo,dnuevo,pi_nuevo]=Api_new(A_pi,aij,d,p_i);
   total_count=total_count+1;
   if Anuevo < A_pi        %Here we decide if Anuevo is better than A_pi
      A_pi=Anuevo;
      d=dnuevo;
      p_i=pi_nuevo;
      max=0;
   else
      max=max+1;
   end
end
new_pi=p_i;
Api_min=A_pi;

/*-------------------------------------------------------------------*/

function A_pi=Api(aij,d,p_i)
%A_pi is the expected waiting time for all clients
%who wish to access one or two files
```

```
n=length(p_i);
A_pi=0;

for j=1:n
    for i=1:j
      A_pi=A_pi+aij(i,j)*(d(i,j)-((d(i,j))^2)/n);
    end
end

A_pi=A_pi+(n/2)+1;

/*-----------------------------------------------------------------------*/

function [Anuevo,d,pi_nuevo]=Api_new(A,aij,dist_ij,p_i)
%Returns the new value of A_pi and the modified matrix of distances

n=length(p_i);
aux=randperm(n);
k=aux(1);       %k and L are generated for doing the swap process
L=aux(2);

k1=p_i(2,k)+1;
L1=p_i(2,L)+1;

if k1 > L1
   k1=L1;
   L1=p_i(2,aux(1))+1;
end

clear aux;
sums=Api_sum(aij,dist_ij,n,k1,L1);
sums=-sums;

%Here we swap the columns and
%(The swap is between k and L)
% Also the swap of the elements of p_i
[pi_nuevo,d]=swap(k,L,dist_ij,p_i,n);

sums=sums+Api_sum(aij,d,n,k1,L1);

Anuevo=A+sums;

/*-----------------------------------------------------------------------*/

function suma=Api_sum(aij,dist_ij,n,k,L)
%Does the sum of the rows and columns specified by k and L
%this sum is the sum of the elements of formula A_pi

suma=0;

for j=1:k-1
     suma=suma+aij(k,j)*(dist_ij(k,j)-(dist_ij(k,j)^2)/n);
end

for j=1:L-1
     suma=suma+aij(L,j)*(dist_ij(L,j)-(dist_ij(L,j)^2)/n);
```

```
end

for i=k+1:n
      suma=suma+aij(i,k)*(dist_ij(i,k)-(dist_ij(i,k)^2)/n);
end

for i=L+1:n
      suma=suma+aij(i,L)*(dist_ij(i,L)-(dist_ij(i,L)^2)/n);
end

suma=suma-aij(L,k)*(dist_ij(L,k)-(dist_ij(L,k)^2)/n);

/*-----------------------------------------------------------------------*/

function [pi_nuevo,d]=swap(k,L,dist_ij,p_i,n)

pi_nuevo=p_i;
d=dist_ij;
pi_nuevo(2,k)=p_i(2,L);
pi_nuevo(2,L)=p_i(2,k);

k=pi_nuevo(2,k)+1;
L=pi_nuevo(2,L)+1;

for i=1:n
   temp=d(k,i);
   d(k,i)=d(L,i);
   d(L,i)=temp;
end


for j=1:n
   temp=d(j,k);
   d(j,k)=d(j,L);
   d(j,L)=temp;
end

clear temp;

/*-----------------------------------------------------------------------*/

function [best_pi,Api_min,time]=Rand_And(total,n,aij)
%choose the best permutation of total random permutations

tic;
best_pi=genera_pi(n);
s=sigm(best_pi);
d dij(s);
Api_min=Api(aij,d,best_pi);

for i=2:total
   p=genera_pi(n);
   s=sigm(p);
   d=dij(s);
   A_pi=Api(aij,d,p);
   if A_pi < Api_min
      Api_min A_p ;
```

```
      best_pi=p;
   end
end
time=toc;

/*-----------------------------------------------------------------------*/

function [best_pi,total_count,Bpi_min,time]=Or_t(t,n,bij)
%OR t gives the best pi permutation of t
%permutations resulting from applying the hill
%climbing algorithm to t random permutations
%the permutations are of size n.
tic;
p=genera_pi(n);
s=sigm(p);
d=dij(s);
[best_pi,total_count,Bpi_min]=Hill_Or(bij,d,p);

if t > 1
   for i=2:t
      p=genera_pi(n);
      s=sigm(p);
      d=dij(s);
      [new_pi_temp,tc,Bpim_temp]=Hill_Or(bij,d,p);
      total_count=total_count+tc;
      if Bpim_temp < Bpi_min
         best_pi=new_pi_temp;
         Bpi_min=Bpim_temp;
      end
   end
end
time=toc;

/*-----------------------------------------------------------------------*/

function [new_pi,total_count,Bpi_min]=Hill_Or(bij,d,p_i)
% Hill Climbing for finding a local minimum for
% a given permutation in the OR problem.

B_pi=Bpi(bij,d,p_i);
total_count=0;  % number of permutations for which
                % fitness function was evaluated.
max=0;

while max < 30
   [Bnuevo,dnuevo,pi_nuevo]=Bpi_new(B_pi,bij,d,p_i);
   total_count total_count+1;
   if Bnuevo < B_pi
      B_pi=Bnuevo;
      d=dnuevo;
      p_i=pi_nuevo;
      max=0;
   else
      max=max+1;
   end
end
```

```
new_pi=p_i;
Bpi_min=B_pi;

/*-----------------------------------------------------------------------*/

function B_pi=Bpi(bij,d,p_i)
%B_pi is the expected waiting time for all clients
%who wish to access one or one out of two files
n=length(p_i);
B_pi=0;
for j=1:n
    for i=1:j
      B_pi=B_pi+bij(i,j)*(d(i,j)-((d(i,j))^2)/n);
    end
end
B_pi=(n/2)+1-B_pi;

/*-----------------------------------------------------------------------*/

function [Bnuevo,d,pi_nuevo]=Bpi_new(B,bij,dist_ij,p_i)
%Returns the new value of B_pi and the modified matrix of distances

n=length(p_i);
aux=randperm(n);
k=aux(1);        %k and L are generated for doing the swap process
L=aux(2);

k1=p_i(2,k)+1;
L1=p_i(2,L)+1;

if k1 > L1
   k1=L1;
   L1=p_i(2,aux(1))+1;
end

clear aux;
sums=Bpi_sum(bij,dist_ij,n,k1,L1);

%Here we swap the columns and
%(The swap is between k and L)
%Also the swap of the elements of p_i

[pi_nuevo,d]=swap(k,L,dist_ij,p_i,n);

sums=sums-Bpi_sum(bij,d,n,k1,L1);

Bnuevo=B+sums;

/*-----------------------------------------------------------------------*/

function s=Bpi_sum(bij,dist_ij,n,k,L)
% Returns the sum of the rows and columns specified by k and L
% this sum is the sum of the elements of formula B_pi

s=0;

for j=1:k-1
```

```
   s=s+bij(k,j)*(dist_ij(k,j)-(dist_ij(k,j)^2)/n);
end

for j=1:L-1
   s=s+bij(L,j)*(dist_ij(L,j)-(dist_ij(L,j)^2)/n);
end

for i=k+1:n
   s=s+bij(i,k)*(dist_ij(i,k)-(dist_ij(i,k)^2)/n);
end

for i=L+1:n
   s=s+bij(i,L)*(dist_ij(i,L)-(dist_ij(i,L)^2)/n);
end

s=s-bij(L,k)*(dist_ij(L,k)-(dist_ij(L,k)^2)/n);

/*-----------------------------------------------------------------------*/

function [best_pi,Bpi_min,time]=Rand_Or(total,n,bij)
%choose the best permutation of total random permutations
tic;
best_pi=genera_pi(n);
s=sigm(best_pi);
d=dij(s);
Bpi_min=Bpi(bij,d,best_pi);

for i=2:total
   p=genera_pi(n);
   s=sigm(p);
   d=dij(s);
   B_pi=Bpi(bij,d,p);
   if B_pi < Bpi_min
      Bpi_min=B_pi;
      best_pi=p;
   end
end
time=toc;

/*-----------------------------------------------------------------------*/

function [best_pi,total_count,Cpi_min,time]=Imply_t(t,n,cij)
%Imply t gives the best pi permutation of t
%permutations resulting from applying the hill
%climbing algorithm to t random permutations
%the permutations are of size n.
tic;
p=genera_pi(n);
s=sigm(p);
dd ddij(s);
[best_pi,total_count,Cpi_min]=Hill_Imply(cij,dd,p);

if t > 1
   for i=2:t
      p genera_pi(n);
      s sigm(p);
      dd ddij(s);
```

```
      [new_pi_temp,tc,Cpim_temp]=Hill_Imply(cij,dd,p);
      total_count=total_count+tc;
      if Cpim_temp < Cpi_min
         best_pi=new_pi_temp;
         Cpi_min=Cpim_temp;
      end
   end
end
time=toc;

/*-------------------------------------------------------------------*/

function dd=ddij(sig)
% Gives the matrix of directed distances between all the files
% of the schedule pi

n=length(sig);
dd=zeros(n,n);

for i=1:n
   for j=1:n
      dd(i,j)=dir_distance(i,j,sig);
   end
end

/*-------------------------------------------------------------------*/

function ddij=dir_distance(i,j,sig)
%Gives the directed distance between files
%Fi end Fj in the infinite cyclic schedule pi
%It's the number of slots between a
%broadcast of Fi and the next broadcast of Fj

ddij=mod(sig(2,j)-sig(2,i),length(sig));

/*-------------------------------------------------------------------*/

function [new_pi,total_count,Cpi_min]=Hill_Imply(cij,dd,p_i)
% Hill Climbing for finding a local minimum for
% a given permutation in the IMPLY problem.

C_pi=Cpi(cij,dd,p_i);
total_count=0;% number of permutations for which
              % fitness function was evaluated.
max=0;

while max < 30
   [Cnuevo,ddnuevo,pi_nuevo]=Cpi_new(C_pi,cij,dd,p_i);
   total_count=total_count+1;
   if Cnuevo < C_pi
      C_pi=Cnuevo;
      dd=ddnuevo;
      p_i=pi_nuevo;
      max=0;
   else
      max=max+1;
   end
```

```
end

new_pi=p_i;
Cpi_min=C_pi;

/*-------------------------------------------------------------------*/

function C_pi=Cpi(cij,dd,p_i)
%C_pi is the expected waiting time for all clients
%who wish to access one file or two files in a certain order

n=length(p_i);
C_pi=0;

for i=1:n
    for j=1:n
      C_pi=C_pi+cij(i,j)*dd(i,j);
    end
end

C_pi=(n/2)+1+C_pi;

/*-------------------------------------------------------------------*/

function [Cnuevo,dd,pi_nuevo]=Cpi_new(C,cij,ddist_ij,p_i)
% Returns the new value of C_pi and the modified matrix of direct distances

n=length(p_i);
aux=randperm(n);
k=aux(1);        %k and L are generated for doing the swap process
L=aux(2);

k1=p_i(2,k)+1;
L1=p_i(2,L)+1;

if k1 > L1
   k1=L1;
   L1=p_i(2,aux(1))+1;
end

clear aux;

sums=Cpi_sum(cij,ddist_ij,n,k1,L1);
sums=-sums;

%Here we swap the columns and
%(The swap is between k and L)
% Also the swap of the elements of p_i
[pi_nuevo,dd]=swap(k,L,ddist_ij,p_i,n);

sums=sums+Cpi_sum(cij,dd,n,k1,L1);

Cnuevo=C+sums;

/*-------------------------------------------------------------------*/

function s=Cpi_sum(cij,ddist_ij,n,k,L)
```

```
%Does the sum of the rows and columns specified by k and L
%this sum is the sum of the elements of formula C'pi

s=0;

for j=1:n
   s=s+cij(k,j)*ddist_ij(k,j);
end

for j=1:n
   s=s+cij(L,j)*ddist_ij(L,j);
end

for i=1:n
   s=s+cij(i,k)*ddist_ij(i,k);
end

for i=1:n
   s=s+cij(i,L)*ddist_ij(i,L);
end

s=s-cij(L,k)*ddist_ij(L,k);
s=s-cij(k,L)*ddist_ij(k,L);

/*-------------------------------------------------------------------------*/

function [best_pi,Cpi_min,time]=Rand_Imply(total,n,cij)
%choose the best permutation of total random permutations

tic;
best_pi=genera_pi(n);
s=sigm(best_pi);
dd=ddij(s);
Cpi_min=Cpi(cij,dd,best_pi);

for i=2:total
   p=genera_pi(n);
   s=sigm(p);
   dd=ddij(s);
   C_pi=Cpi(cij,dd,p);
   if C_pi < Cpi_min
      Cpi_min=C_pi;
      best_pi=p;
   end
end

time=toc;

/*-------------------------------------------------------------------------*/

function [new_pi,Api_array]=Hill_And_m(aij,n,total)
% Hill Climbing for finding a local maximum for
% a given permutation in the AND problem.

i=1;
Api_array=zeros(2,total);
tic;
```

```
p=genera_pi(n);
s=sigm(p);
d=dij(s);
A_pi=Api(aij,d,p);
Api_array(1,i)=A_pi;
time=toc;
tic;
Api_array(2,i)=time;


for i=2:total
   [Anuevo,dnuevo,pi_nuevo]=Api_new(A_pi,aij,d,p);
   Api_array(1,i)=Anuevo;
   aux=toc;
   tic;
   time=time+aux;
   Api_array(2,i)=time;
   if Anuevo < A_pi        %Here we decide if Anuevo is better than A_pi
      p=pi_nuevo;
      d=dnuevo;
      A_pi=Anuevo;
   end
end
new_pi=p;

/*-----------------------------------------------------------------*/

function [best_pi,Api_array]=Rand_And_m(aij,n,total)
%choose the best permutation of total random permutations

Api_array=zeros(2,total);
tic;
best_pi=genera_pi(n);
s=sigm(best_pi);
d=dij(s);
Api_min=Api(aij,d,best_pi);
Api_array(1,1)=Api_min;
time=toc;
tic;
Api_array(2,1)=time;

for i=2:total
   p=genera_pi(n);
   s=sigm(p);
   d=dij(s);
   A_pi=Api(aij,d,p);
   aux=toc;
   tic;
   time=time+aux;
   Api_array(1,i)=A_pi;
   Api_array(2,i)=time;
   if A_pi < Api_min
      best_pi=p;
   end
end

/*-----------------------------------------------------------------*/
```

```
function [new_pi,Bpi_array]=Hill_Or_m(bij,n,total)
% Hill Climbing for finding a local minimum for
% a given permutation in the AND problem.

i=1;
Bpi_array=zeros(2,total);
tic;
p=genera_pi(n);
s=sigm(p);
d=dij(s);
B_pi=Bpi(bij,d,p);
Bpi_array(1,i)=B_pi;
time=toc;
tic;
Bpi_array(2,i)=time;


for i=2:total
   [Bnuevo,dnuevo,pi_nuevo]=Bpi_new(B_pi,bij,d,p);
   Bpi_array(1,i)=Bnuevo;
   aux=toc;
   tic;
   time=time+aux;
   Bpi_array(2,i)=time;
   if Bnuevo < B_pi        %Here we decide if Anuevo is better than A_pi
      p=pi_nuevo;
      d=dnuevo;
      B_pi=Bnuevo;
   end
end
new_pi=p;

/*-----------------------------------------------------------------------*/

function [best_pi,Bpi_array]=Rand_Or_m(bij,n,total)
%choose the best permutation of total random permutations

Bpi_array=zeros(2,total);
tic;
best_pi=genera_pi(n);
s=sigm(best_pi);
d=dij(s);
Bpi_min=Bpi(bij,d,best_pi);
Bpi_array(1,1)=Bpi_min;
time=toc;
tic;
Bpi_array(2,1)=time;

for i=2:total
   p=genera_pi(n);
   s=sigm(p);
   d=dij(s);
   B_pi=Bpi(bij,d,p);
   aux=toc;
   tic;
   time=time+aux;
```

```
   Bpi_array(1,i)=B_pi;
   Bpi_array(2,i)=time;
   if B_pi < Bpi_min
      best_pi=p;
   end
end

/*-----------------------------------------------------------------------*/

function [new_pi,Cpi_array]=Hill_Imply_m(cij,n,total)
% Hill Climbing for finding a local minimum for
% a given permutation in the AND problem.

i=1;
Cpi_array=zeros(2,total);
tic;
p=genera_pi(n);
s=sigm(p);
dd=ddij(s);
C_pi=Cpi(cij,dd,p);
Cpi_array(1,i)=C_pi;
time=toc;
tic;
Cpi_array(2,i)=time;


for i=2:total
   [Cnuevo,ddnuevo,pi_nuevo]=Cpi_new(C_pi,cij,dd,p);
   Cpi_array(1,i)=Cnuevo;
   aux=toc;
   tic;
   time=time+aux;
   Cpi_array(2,i)=time;
   if Cnuevo < C_pi        %Here we decide if Anuevo is better than A_pi
      p=pi_nuevo;
      dd=ddnuevo;
      C_pi=Cnuevo;
   end
end

new_pi=p;

/*-----------------------------------------------------------------------*/

function [best_pi,Cpi_array]=Rand_Imply_m(cij,n,total)
%choose the best permutation of total random permutations

Cpi_array=zeros(2,total);
tic;
best_pi=genera_pi(n);
s=sigm(best_pi);
dd=ddij(s);
Cpi_min=Cpi(cij,dd,best_pi);
Cpi_array(1,1)=Cpi_min;
time=toc;
tic;
Cpi_array(2,1)=time;
```

```
for i=2:total
   p=genera_pi(n);
   s=sigm(p);
   dd=ddij(s);
   C_pi=Cpi(cij,dd,p);
   aux=toc;
   tic;
   time=time+aux;
   Cpi_array(1,i)=C_pi;
   Cpi_array(2,i)=time;
   if C_pi < Cpi_min
      best_pi=p;
   end
end

/*-----------------------------------------------------------------------*/

function Variando_ml(n,maxm,files)

probabilts(n,files); %generates aij, bij and cij and save it in the file
probs
load('probs'); % loads the variables aij, bij and cij

diary probabilities.txt
   aij
   bij
   cij
diary off;

diary files.txt
   files
diary off;

[And_Hill_pi,And_Hill_array]=Hill_And_m(aij,n,maxm);
[And_Rand_pi,And_Rand_array]=Rand_And_m(aij,n,maxm);

[Or_Hill_pi,Or_Hill_array]=Hill_Or_m(bij,n,maxm);
[Or_Rand_pi,Or_Rand_array]=Rand_Or_m(bij,n,maxm);

[Imply_Hill_pi,Imply_Hill_array]=Hill_Imply_m(cij,n,maxm);
[Imply_Rand_pi,Imply_Rand_array]=Rand_Imply_m(cij,n,maxm);

diary Hill_And.txt
   And_Hill_pi
   And_Hill_array
diary off;

diary Rand_And.txt
   And_Rand_pi
   And_Rand_array
diary off;

diary Hill_Or.txt
   Or_Hill_pi
   Or_Hill_array
diary off;
```

```
diary Rand_Or.txt
   Or_Rand_pi
   Or_Rand_array
diary off;

diary Hill_Imply.txt
   Imply_Hill_pi
   Imply_Hill_array
diary off;

diary Rand_Imply.txt
   Imply_Rand_pi
   Imply_Rand_array
diary off;

% Graphs for the AND case
figure;
plot(And_Hill_array(1,:));
title('Algoritmo de hill climbing para el caso AND');
xlabel('N° de permutaciones evaluadas');
ylabel('Tiempo de acceso esperado en segundos');
grid on;
figure;
plot(And_Hill_array(2,:));
xlabel('N° de permutaciones evaluadas');
ylabel('Tiempo de procesamiento en segundos');
grid on;
figure;
plot(And_Rand_array(1,:));
title('Algoritmo aleatorio para el caso AND');
xlabel('N° de permutaciones evaluadas');
ylabel('Tiempo de acceso esperado en segundos');
grid on;
figure;
plot(And_Rand_array(2,:));
xlabel('N° de permutaciones evaluadas');
ylabel('Tiempo de procesamiento en segundos');
grid on;

% Graphs for the OR case
figure;
plot(Or_Hill_array(1,:));
title('Algoritmo de hill climbing para el caso OR');
xlabel('N° de permutaciones evaluadas');
ylabel('Tiempo de acceso esperado en segundos');
grid on;
figure;
plot(Or_Hill_array(2,:));
xlabel('N° de permutaciones evaluadas');
ylabel('Tiempo de procesamiento en segundos');
grid on;
figure;
plot(Or_Rand_array(1,:));
title('Algoritmo aleatorio para el caso OR');
xlabel('N° de permutaciones evaluadas');
ylabel('Tiempo de acceso esperado en segundos');
```

```
grid on;
figure;
plot(Or_Rand_array(2,:));
xlabel('N° de permutaciones evaluadas');
ylabel('Tiempo de procesamiento en segundos');
grid on;

% Graphs for the IMPLY case
figure;
plot(Imply_Hill_array(1,:));
title('Algoritmo de hill climbing para el caso IMPLY');
xlabel('N° de permutaciones evaluadas');
ylabel('Tiempo de acceso esperado en segundos');
grid on;
figure;
plot(Imply_Hill_array(2,:));
xlabel('N° de permutaciones evaluadas');
ylabel('Tiempo de procesamiento en segundos');
grid on;
figure;
plot(Imply_Rand_array(1,:));
title('Algoritmo aleatorio para el caso IMPLY');
xlabel('N° de permutaciones evaluadas');
ylabel('Tiempo de acceso esperado en segundos');
grid on;
figure;
plot(Imply_Rand_array(2,:));
xlabel('N° de permutaciones evaluadas');
ylabel('Tiempo de procesamiento en segundos');
grid on;


/*-----------------------------------------------------------------------*/

function probabilts(n,files)
%Generates aij, bij and cij, they are stored in file probs.mat

aij=and_or(n,files);
bij=aij;
cij=imply(n,files);

save probs aij bij cij;

/*-----------------------------------------------------------------------*/

function [xij,a]=and_or(n,files)
%Returns the matrix of probabilities for
%the AND and OR cases.

xij=prob(n);
%generates a matrix of random probabilities with size[n,n]

a ones(n,n);
lar=length(files);

for p=1:lar-1
   for i=files(1,p)+1:files(1,p+1)
```

```
      for j=1:files(1,p)
         a(i,j)=1e-6;
         a(j,i)=a(i,j);
      end
   end
end

for i=1:n
    for j=1:n
        if xij(i,j) ~= xij(j,i)
           xij(i,j)=(xij(i,j)+xij(j,i))/2;
           %This is done because xij=xji
           xij(j,i)=xij(i,j);
        end
     end
end

xij=xij.*a;

xij=xij/(sum(sum(tril(xij))));

/*-----------------------------------------------------------------------*/

function [xij,a]=imply(n,files)
%Returns the matrix of probabilities for
%the AND and OR cases.

xij=abs(randn(n));
%generates a matrix of random probabilities with size[n,n]

a=ones(n,n);
lar=length(files);

for p=1:lar-1
   for i=files(1,p)+1:files(1,p+1)
      for j=1:files(1,p)
         a(i,j)=1e-6; %xij(i,j)/100; ?=0
         a(j,i)=a(i,j);
      end
   end
end

xij=xij.*a;
xij=xij/sum(sum(xij));

/*-----------------------------------------------------------------------*/

function x=prob(n)
%generates a matrix of random probabilities with size[n,n]

x=abs(randn(n));
x=x/sum(sum(x));
/*-----------------------------------------------------------------------*/
```