

Universidad de Buenos Aires
Facultad de Ciencias Exactas y Naturales

Tesis de Licenciatura
Ciencias de la Computación

***Nuevas cotas para el problema de
Paginación On-line de Múltiples Procesos
(MTP)***

Darío Germán Robak

1997

Director: Dr. Esteban Feuerstein

A mis padres.

AGRADECIMIENTOS

A Alejandro Strejilevich de Loma, por su muy importante ayuda y compañerismo.

A mis compañeros Emilio Platzer, Gustavo Vulcano, Pablo Coll, Guillermo Durán, Min Chih Lin y Paula Zabala, por el apoyo constante durante los años que compartimos.

A mi director de tesis Esteban Feuerstein, por su confianza y amistad.

A los profesores Irene Loiseau e Isabel Méndez, por sus sugerencias y buenas ideas.

A mis padres. A Gaby, Goyo y Romi. A Pablo y Michelle.

A Fedra.

A todos ellos,
gracias.

Darío G. Robak.

Tabla de contenido

PREFACIO	6
1. INTRODUCCIÓN	8
1.1 GENERALIDADES SOBRE ALGORITMOS ON-LINE Y COMPETITIVIDAD	8
1.2 PAGINACIÓN DE MEMORIA	10
1.3 PAGINACIÓN DE MEMORIA DE MÚLTIPLES PROCESOS (MTP)	11
1.4 DIFERENTES VARIANTES DEL PROBLEMA MTP	12
1.5 COTAS CONOCIDAS	12
1.5.1 <i>Paginación estándar</i>	13
1.5.2 <i>FMTP: MTP finito</i>	13
1.5.3 <i>IMTP: MTP infinito</i>	13
1.5.4 <i>t-Fair-IMTP: t-Fair-MTP infinito</i>	14
1.6 RELACIÓN CON OTROS PROBLEMAS	14
1.7 NOTACIÓN Y CONVENCIONES	14
1.8 RESUMEN DE RESULTADOS	15
1.9 IMPORTANCIA PRÁCTICA	16
2. RESTRICCIONES SOBRE ALGORITMOS E INSTANCIAS	17
2.1 INTRODUCCIÓN	17
2.2 ALGORITMOS LAZY	17
2.3 SERVICIO DE PÁGINAS EN LA CACHE	18
2.4 INSTANCIAS SUAVES Y DURAS	20
2.5 INSTANCIAS CON COSTO NO ACOTADO	21
3. CACHE DE TAMAÑO UNO	23
3.1 INTRODUCCIÓN	23
3.2 UN ALGORITMO OFF-LINE ÓPTIMO	23
3.2.1 <i>Dos Secuencias</i>	23
3.2.2 <i>Cualquier número de secuencias</i>	25
3.3 ALGORITMOS W-COMPETITIVOS	28
4. CACHE DE TAMAÑO UNO Y DOS SECUENCIAS	29
4.1 INTRODUCCIÓN	29
4.2 RELACIÓN ENTRE LAS LONGITUDES DE AMBAS SECUENCIAS	30
4.3 MÁXIMA CANTIDAD DE PEDIDOS NO APAREADOS	31
4.4 ALGORITMOS 2-COMPETITIVOS	32
4.5 NUEVA COTA INFERIOR	38
4.5.1 <i>Generación de las instancias</i>	38
4.5.2 <i>Instancias de costo arbitrariamente grande</i>	41
4.5.3 <i>Análisis de la instancia generada</i>	41
4.5.4 <i>Algunos árboles</i>	45
4.5.5 <i>Programa para analizar el árbol generado</i>	49
4.5.6 <i>Cantidad de páginas necesarias</i>	51
5. MULTI-THREADED PAGING SEMI-ONLINE	52
5.1 INTRODUCCIÓN	52
5.2 DEFINICIÓN DEL PROBLEMA	52
5.3 COTA INFERIOR	53
6. CACHE DE TAMAÑO UNO Y VARIAS SECUENCIAS	58
6.1 INTRODUCCIÓN	58
6.2 COTAS SUPERIORES	59

6.2.1 Un algoritmo on-line para varias secuencias.....	59
6.2.2 Un algoritmo Divide And Conquer.....	62
6.3 MEJORA DE LA COTA INFERIOR.....	65
6.3.1 Adaptación de resultados previos.....	65
6.3.2 Generación de la instancia.....	66
6.3.3 Análisis de la instancia.....	68
6.3.4 Instancias de costo arbitrariamente grande.....	68
6.3.5 Adaptación a cualquier cantidad de secuencias.....	69
7. UN PROBLEMA OFF-LINE	71
7.1 INTRODUCCIÓN.....	71
7.2 DEFINICIÓN DEL PROBLEMA	72
7.3 COTAS PARA EL CASO INFINITO	73
7.4 COTAS PARA EL CASO FINITO.....	77
7.4.1 Estrategias.....	80
7.4.2 Cota superior.....	82
8. CACHÉ DE TAMAÑO ARBITRARIO	86
8.1 INTRODUCCIÓN.....	86
8.2 ALGORITMOS KW-COMPETITIVOS PARA MTP.....	86
8.3 GENERALIZACIÓN DE COTAS INFERIORES CON CACHÉ DE TAMAÑO UNO.....	87
8.4 MEJORA DE LA COTA INFERIOR.....	90
9. ALGORITMOS ALEATORIOS.....	91
9.1 INTRODUCCIÓN.....	91
9.2 COMPETITIVIDAD Y ALGORITMOS ALEATORIOS.....	91
9.3 COTA INFERIOR	92
10. CONCLUSIONES	95
11. APÉNDICE A	97
12. REFERENCIAS.....	100

Prefacio

Esta tesis contiene los resultados de mi trabajo de investigación realizado en el Departamento de Computación de la Facultad de Ciencias Exactas y Naturales, en la Universidad de Buenos Aires, durante los años 1996 y 1997.

El trabajo analiza el problema Multi-Threaded Paging (MTP – Paginación de Múltiples procesos), el cual es una extensión del problema de Paginación de memoria donde se presentan más de una secuencia de pedidos a un algoritmo, que debe decidir no sólo cómo servir estos pedidos sino también qué pedidos servir. La cantidad de page-faults incurridos en servir todos los pedidos va a depender tanto de las páginas desalojadas como del orden en que se sirven los pedidos de las distintas secuencias. Este problema, propuesto por E. Feuerstein y A. Strejilevich de Loma ([1], [2]), es de importancia práctica en la administración de memoria en sistemas multitarea.

En esta tesis se estudia el problema MTP en general, proveyendo cotas a la competitividad de diferentes algoritmos on-line. También se estudian diferentes casos particulares de este problema, obteniendo mejores resultados que en el caso general. Finalmente se propone una extensión de MTP que permite modelar con mayor fidelidad problemas de secuenciamiento de tareas, y se encuentran cotas para un caso particular de esta extensión.

Este trabajo está organizado en diez capítulos. El primero es introductorio y presenta los conceptos básicos sobre algoritmos on-line y análisis de competitividad.

En el capítulo 2 se introducen restricciones sobre las instancias y los algoritmos que se analizarán en el resto de la tesis y se muestra que con estas restricciones no se pierde generalidad en los resultados obtenidos.

En los capítulos 3, 4, 5 y 6 se analiza el caso particular del problema MTP en el cual la caché es de tamaño uno, presentando algoritmos óptimos off-line y cotas para la competitividad de algoritmos on-line. El capítulo 3 presenta resultados generales sobre este caso particular. El capítulo 4 analiza el caso aún más particular en el cual la caché es de tamaño uno y se tienen únicamente dos secuencias de pedidos. En el capítulo 5 se presenta una extensión del problema MTP que llamamos MTP Semi-online, y se presentan cotas para el caso en que la caché es de tamaño uno y se tienen dos secuencias de pedidos. En el capítulo 6 se presentan cotas para el problema MTP para el caso en que se tienen más de 2 secuencias de pedidos.

En el capítulo 7 se analiza un problema off-line relacionado con MTP. Los resultados de este capítulo son útiles en el capítulo siguiente al obtener cotas para MTP en su forma general.

En el capítulo 8 se analiza el problema MTP con caché de cualquier tamaño y cualquier número de secuencias de pedidos, obteniéndose cotas de competitividad para los algoritmos on-line.

El capítulo 9 se dedica al problema MTP cuando se consideran algoritmos on-line aleatorios, presentando una nueva cota inferior.

Finalmente, el capítulo 10 presenta un resumen final de la tesis, conclusiones y una visión general sobre futuras direcciones de investigación.

1. Introducción

1.1 Generalidades sobre algoritmos On-line y Competitividad

Durante los últimos años se ha dedicado una considerable atención al análisis de competitividad de algoritmos on-line. Informalmente, un algoritmo on-line es aquel que debe tomar decisiones antes de conocer completamente los datos del problema. Estas decisiones son tomadas teniendo en cuenta los requerimientos pasados y suponiendo, en general, que los requerimientos futuros se asemejarán a los previos.

Un algoritmo on-line puede ser visto como un algoritmo que recibe una secuencia de pedidos y realiza una acción inmediata en respuesta a cada uno de ellos. Algoritmos de este tipo se usan en todas aquellas situaciones en las cuales es necesario tomar decisiones y asignar recursos sin conocimiento del futuro. En contraposición, se dice que un algoritmo es off-line si conoce completamente los datos del problema a resolver desde el principio. Ejemplos de algoritmos on-line aparecen en la administración de estructuras de datos dinámicas, aplicaciones interactivas, planificación de movimientos (robots, cabezales de discos rígidos, ascensores), juegos (Tetris), etc.

La naturaleza on-line de estos algoritmos puede ocasionar una pérdida de performance con respecto a los resultados que podrían obtenerse con un conocimiento *a priori* de los datos del problema. Por lo tanto, surge la necesidad de establecer un criterio que permita evaluar la calidad de diferentes algoritmos on-line. Este criterio no es obvio y puede basarse en distintos tipos de análisis. El análisis típico en algoritmos off-line del *peor caso* no es siempre conveniente pues a veces no logra distinguir entre diferentes algoritmos on-line con distinta performance.

Otra alternativa consiste en asumir un determinado modelo probabilístico para los datos del problema. En ese caso un algoritmo puede ser considerado óptimo si escoge sus acciones de manera tal que se minimice la esperanza de su costo. Ese es el enfoque tradicionalmente seguido por algunos sectores, en particular aquellos relacionados con las teorías de Juegos y Control Estocásticos. La principal desventaja es que requiere conocer (o asumir) la distribución de probabilidad de los datos de entrada.

Para resolver estos inconvenientes aparece la noción de *Análisis de Competitividad*, introducida en 1985 por Sleator y Tarjan[10]. Según esta metodología se compara un algoritmo on-line con el algoritmo óptimo que conoce completamente la instancia de datos del problema. Se utiliza el algoritmo óptimo off-line como un punto de partida, y se comparan los distintos algoritmos on-line con él.

Según este punto de vista, la calidad de un algoritmo on-line se mide por la máxima razón entre el costo del algoritmo on-line con el costo incurrido por el algoritmo óptimo

off-line en la misma instancia de datos. La máxima razón se calcula sobre el universo de instancias del problema.

Formalmente, llamando $C_A(I)$ al costo del algoritmo on-line A sobre la instancia I y $C_o(I)$ al costo óptimo off-line sobre la misma instancia, se dice que el algoritmo A es *c-competitivo* si existe un b tal que para toda instancia I del problema:

$$C_A(I) \leq c C_o(I) + b$$

La constante c se llama factor de competitividad. Se dice que un algoritmo on-line A es competitivo si es c -competitivo para alguna constante c . Esto es útil pues para algunos problemas no existen algoritmos on-line c -competitivos, para ningún valor de c . Se dice también que la competitividad de un algoritmo on-line A es el ínfimo de los valores c tal que A es c -competitivo.

Este tipo de análisis evita los inconvenientes del análisis del caso promedio y permite diferenciar algoritmos de distinta performance. A partir del trabajo de Sleator y Tarjan, una gran cantidad de investigadores se ha dedicado al estudio del tema, tanto a la solución eficiente de problemas on-line particulares como a la obtención de herramientas generales y metodologías de análisis más refinadas.

Los algoritmos competitivos tienen un cierto sentido de aprendizaje, pues aprovechan la estructura de los datos conocidos para procesar en forma eficiente la instancia de datos completa.

Es clásico analizar el problema como un juego entre un jugador on-line y un adversario que elige la instancia de datos. El adversario intenta maximizar la razón entre los costos del jugador y el suyo. Esta representación permite obtener dos tipos de resultados: Cotas inferiores y superiores. Las cotas inferiores indican que no existen algoritmos c -competitivos con c menor que el valor de la cota. En general, para todo algoritmo on-line se proveen instancias de datos tal que la razón de costos sea mayor o igual a la cota. Estas instancias se pueden obtener a partir de la estrategia de juego del adversario en el juego hipotético.

Las cotas superiores se obtienen generalmente presentando un algoritmo on-line cuya competitividad es igual a la cota. Estos algoritmos se pueden construir indicando cómo debe responder el jugador on-line a las jugadas del adversario. De esta manera, se puede mostrar que para cualquier acción del adversario el algoritmo obtiene una razón de costos menor o igual a la cota superior.

Un objetivo del estudio de la competitividad de los algoritmos es lograr cotas inferiores y superiores iguales, y hallar un algoritmo cuya competitividad sea igual a estas cotas.

1.2 Paginación de memoria

En los sistemas operativos que ofrecen memoria virtual se utiliza una técnica que permite crear la ilusión de un espacio de memoria mayor a la cantidad de memoria física presente en el sistema. Esta técnica permite ejecutar programas con grandes requerimientos de memoria, en forma independiente del tamaño exacto de la memoria física.

Para crear esta ilusión se modela la memoria en dos niveles. Uno de ellos presenta limitada capacidad y tiempo de acceso corto. Este nivel es llamado la caché y en general se refiere a la memoria principal del sistema. El segundo nivel presenta tiempo de acceso mucho mayor pero una capacidad potencialmente ilimitada. Este nivel es llamado memoria secundaria y generalmente se refiere a medios masivos de almacenamiento, como por ejemplo los discos rígidos.

La memoria se divide en bloques de tamaño fijo llamados páginas, de las cuales la caché puede contener a lo sumo un número máximo que llamaremos k . En cada momento, sólo una parte del espacio de direcciones se encuentra en la caché, mientras que el resto reside en la memoria secundaria. Como únicamente la parte de la memoria que se encuentra en la caché es accesible al procesador, se necesita de una estrategia de paginación para decidir qué páginas de la memoria virtual deben encontrarse en la caché.

A medida que el programa se ejecuta, la vecindad de referencias a memoria cambia, haciendo que ciertas zonas de memoria virtual sean traídas del segundo nivel a la caché, y que otras sean desalojadas de la misma con el objeto de liberar espacio. La estrategia de paginación define que páginas deben ser desalojadas al servir nuevos pedidos.

En el estudio de este problema se asume que el costo principal del sistema se produce en los accesos a memoria secundaria, pues son mucho más lentos que los accesos a páginas en la caché. El programa que se está ejecutando se modela como una secuencia de pedidos a páginas de memoria. El sistema debe servir estos pedidos en forma secuencial. No es posible servir un pedido a una página de memoria si no se han servido previamente todos los anteriores.

Si la página pedida se encuentra en la caché, el costo de servir este pedido se considera 0. En cambio, si la página debe ser traída de la memoria secundaria, el costo de servir este pedido es 1. En este caso se dice que el sistema incurre en un page-fault (falla de página).

Se buscan estrategias de paginación de memoria que minimicen el costo incurrido al servir una secuencia de pedidos, o sea, el número total de page-faults. Este mínimo puede ser alcanzado conociendo la secuencia entera de pedidos [18]. La estrategia óptima en este caso consiste en desalojar una página de la caché sólo cuando hay un pedido a una página que no se encuentra en la misma, y desalojar la página cuyo pedido es más lejano en la secuencia de pedidos. Esta estrategia es sin duda una estrategia off-line, pues requiere conocer la secuencia entera de pedidos para decidir la página a

desalojar de la caché. Como en general no es posible conocer la secuencia de pedidos de antemano, se desarrollaron estrategias basadas en la localidad de referencia del programa. Estas estrategias son algoritmos on-line que deben tomar decisiones sin conocer los próximos pedidos a páginas que hará el programa.

Algunos ejemplos clásicos de algoritmos on-line para paginación de memoria son:

- FIFO (First In First Out – Primero en entrar Primero en salir): Se desaloja la página que ha estado en la caché por más tiempo.
- LRU (Least Recently Used – Menos recientemente usada): Se desaloja la página que no ha sido pedida por más tiempo.
- LIFO (Last In First Out – Ultimo en entrar Primero en salir): Se desaloja la última página cargada en la caché.
- LFU (Least Frequently Used – Menos frecuentemente usada): Se desaloja la página que ha sido menos pedida.

1.3 Paginación de memoria de múltiples procesos (MTP)

El problema Multi-Threaded Paging (MTP – Paginación Multi-Proceso) es una generalización del problema de paginación de memoria. Fue introducido por Feuerstein[1] y Feuerstein-Strejilevich de Loma[2].

En este problema no se considera un solo proceso ejecutándose en el sistema sino un número arbitrario de ellos. Como cada uno realiza pedidos a páginas de memoria, la estrategia de paginación debe ampliarse para decidir no sólo qué página desalojar, sino también qué proceso servir. La cantidad de page-faults incurridos en servir todos los pedidos va a depender tanto de las páginas desalojadas como del orden en que se sirven los pedidos de los distintos procesos.

En este problema cada proceso se modela como una secuencia de pedidos a páginas. El sistema debe servir estos pedidos en forma ordenada en cada secuencia. Un pedido puede ser servido solamente si los pedidos anteriores en su misma secuencia han sido previamente servidos. El costo de servir un pedido es 0 si la página pedida se encuentra en la caché y 1 en caso contrario. La cantidad de procesos se denota con w .

Cuando se analiza el problema MTP on-line, se asume que un algoritmo on-line puede ver solamente un pedido por proceso, precisamente el primer pedido aún no servido del proceso. Recién después de servir un pedido de un proceso, el algoritmo puede ver el siguiente pedido del mismo.

En algunos casos más de un pedido puede ser servido simultáneamente. En este caso el siguiente pedido de cada proceso servido es presentado al algoritmo. El algoritmo debe decidir en cada momento qué pedidos servir y qué páginas desalojar de la caché.

1.4 Diferentes variantes del problema MTP

El problema MTP presenta diferentes variantes. Las más importantes son las siguientes:

- MTP finito (FMTP): En esta variante la cantidad de pedidos de cada secuencia debe ser finita.
- MTP infinito (IMTP): En esta variante las secuencias de pedidos son infinitas.
- MTP sin control de inanición: En esta variante no existen restricciones en el tiempo que puede esperar un proceso para que sean servidos sus pedidos. Esta variante es de alguna manera injusta, pues es posible que un proceso sea detenido mientras se sirven los pedidos de los demás procesos.
- MTP con control de inanición (Fair-MTP / MTP justo): En esta variante existen restricciones en el tiempo que puede esperar un proceso para que sean servidos sus pedidos. Es más importante en el caso infinito pues un proceso podría esperar indefinidamente sin que sea servido ninguno de sus pedidos. El control de inanición puede modelarse de varias maneras:
 1. No se permite que un pedido espere más de t unidades de tiempo antes de ser servido. El tiempo se mide de la forma intuitiva: cada vez que se sirve uno o varios pedidos simultáneos transcurre una unidad de tiempo. Esta variedad se denomina t -Fair-MTP
 2. No se permite que se sirvan más de δ pedidos de un proceso que de otro. Esta variedad se denomina δ -Fair-MTP. Se puede ver que si un algoritmo es δ -Fair según esta definición, entonces es $2\delta(w-1)$ -Fair según la definición anterior.

Las distintas variantes del problema presentan diferentes cotas inferiores y superiores, como se verá en la siguiente sección. Los resultados obtenidos en esta tesis se refieren exclusivamente a la variante MTP finito sin control de inanición, y usaremos las siglas MTP para mencionarla.

1.5 Cotas conocidas

El problema de paginación de memoria y el problema MTP fueron estudiados previamente. En la literatura se encuentran diferentes cotas inferiores y superiores para estos problemas. A continuación veremos un pequeño resumen de las mismas.

Las letras k y w representan el tamaño de la caché y la cantidad de secuencias, respectivamente.

1.5.1 Paginación estándar

Sleator y Tarjan[10] probaron que los algoritmos FIFO y LRU presentados anteriormente son k -competitivos. Esto significa que para toda secuencia de pedidos a páginas, el costo incurrido por estos algoritmos al servirla es a lo sumo k veces el costo óptimo, salvo por una constante independiente de la secuencia. Los algoritmos LIFO y LFU no presentan esta propiedad.

Sleator y Tarjan también probaron que no existe ningún algoritmo on-line cuya competitividad sea menor a k . Por lo tanto, los algoritmos FIFO y LRU son óptimos según esta medida. Sobre la base de estos resultados, las cotas inferior y superior para el problema de paginación estándar son iguales y su valor es k .

1.5.2 FMTP: MTP finito

En los trabajos de Feuerstein[1] y Feuerstein-Strejilevich de Loma[2] se probó que existen algoritmos wk -competitivos para el problema FMTP. Un algoritmo con esta propiedad presentado en dichas publicaciones es AFWF (Alternate Flush When Full – Desalojo alternado cuando se llena). También se probó que ningún algoritmo puede ser c -competitivo con $c < k + 1 - 1/w$.

Estos resultados proveen una cota inferior para el problema FMTP igual a $k + 1 - 1/w$ y una cota superior igual a wk . Notar que la cota inferior está acotada superiormente por $k+1$ mientras que la cota superior es lineal en w .

El objetivo de esta tesis se centra en disminuir la diferencia entre estas cotas.

1.5.3 IMTP: MTP infinito

En [1] y [2] también se analizó el problema IMTP. Se probó que el mismo algoritmo AFWF usado en FMTP es wk -competitivo para IMTP. A diferencia del caso FMTP, en este caso se pudo probar que no pueden existir algoritmos c -competitivos con $c < wk$ para este problema.

Sobre la base de estos resultados, las cotas inferior y superior son iguales y su valor es wk .

1.5.4 t -Fair-IMTP: t -Fair-MTP infinito

En [2] se mostró que no existen algoritmos on-line competitivos para este problema.

1.6 Relación con otros problemas

El problema MTP está relacionado con varios problemas. Algunos ejemplos son:

- Secuenciamiento de tareas: El orden en que MTP sirve los pedidos de las diferentes secuencias puede ser visto como un caso particular de secuenciamiento de tareas.
- Longest Common Subsequence (LCS – Subsecuencia común de longitud máxima): El caso particular de MTP en que la caché es de tamaño uno y dos secuencias es el conocido problema LCS.
- Maximum Sequence Alignment (MSA – Alineación máxima de secuencias): Cuando la caché es de tamaño uno y hay varias secuencias, el problema MTP es similar al problema MSA.
- Merge (combinación) y compresión de datos: Se puede usar LCS para realizar merge de secuencias y compresión de datos. Al ser LCS un caso particular de MTP, se agrega una nueva dimensión de posibilidades.
- Biología Molecular: El problema de apareamiento de secuencias de ADN está relacionado con el problema MTP.

El análisis de competitividad del problema MTP es útil en cada uno de los problemas anteriormente citados.

1.7 Notación y convenciones

En los siguientes capítulos denotaremos con I a una instancia del problema MTP. Esta instancia es la entrada al algoritmo on-line. Las secuencias de la instancia I las denotaremos con $S_1, \dots, S_w$ y sus longitudes con $L_1, \dots, L_w$, respectivamente.

Llamaremos $C_A(I)$ al costo del algoritmo on-line sobre la instancia I y $C_o(I)$ al costo óptimo off-line sobre la misma instancia. Cuando no exista confusión omitiremos la instancia I y usaremos simplemente C_A y C_o .

En varias oportunidades necesitaremos el concepto de instancias en las cuales el costo óptimo off-line es arbitrariamente grande. Para simplificar definimos el *costo de una instancia* I como $C_o(I)$.

Servir una instancia significa servir todos los pedidos de las secuencias que la componen.

Diremos que dos pedidos son iguales si ambos piden la misma página. Finalmente, diremos que un algoritmo *aparea* pedidos cuando sirve los pedidos conjuntamente. Esto es posible sólo cuando los pedidos son iguales.

Para diferenciar los distintos casos en que analizaremos el problema MTP, notaremos $MTP(k, w)$ al problema MTP con caché de tamaño k y w secuencias. En particular, $MTP(1, w)$ se refiere al problema MTP con caché de tamaño uno y w secuencias, mientras que $MTP(1, 2)$ restringe también la cantidad de secuencias a dos. Para referirnos al problema MTP en general usaremos indistintamente MTP y $MTP(k, w)$.

1.8 Resumen de resultados

En los siguientes capítulos presentaremos los resultados obtenidos para el problema MTP finito sin control de inanición. Mostraremos la reducción de la brecha entre las cotas inferior y superior conocidas para este problema.

El análisis se realizó separando el problema en diferentes casos. Esto permitió obtener mejores cotas para casos particulares que las obtenidas para el caso general. Se analizaron algunos problemas relacionados con el problema MTP. También se mostró cómo utilizar resultados obtenidos en casos particulares para obtener resultados para el problema en su forma general..

Los resultados principales son los siguientes:

- Adaptamos un resultado obtenido por Alborzi et al.[4] al problema $MTP(1, w)$ obteniendo una cota inferior igual a $\frac{\lfloor \log w \rfloor}{2} + 1$. La cota conocida anteriormente era $2 - 1/w$.
- En el caso $MTP(1, 2)$ en el que la caché es de tamaño uno y hay dos secuencias, esta cota da un valor igual a $\frac{\lfloor \log 2 \rfloor}{2} + 1 = 1.5$, igual a la cota anterior conocida.

Realizando un análisis para este caso particular pudimos obtener una cota inferior igual a 1.788. También mostramos cómo utilizar cotas superiores obtenidas para MTP(1,2) para obtener cotas superiores para MTP(1,w).

- Analizamos una variante del problema MTP(1,2), en el cual una de las dos secuencias es conocida para el algoritmo on-line. Probamos una cota inferior para este problema igual a la razón áurea.
- Finalmente mostramos cómo adaptar la cota inferior obtenida para MTP(1,w) al caso general MTP(k,w), obteniendo una cota inferior para el problema MTP(k,w) igual a $\left(\frac{\lfloor \log w \rfloor}{2} + 1\right) \frac{k+e/2}{e}$. La cota inferior conocida anteriormente era de $k + 1 - 1/w$. La cota actual crece logarítmicamente con respecto a w y linealmente con respecto a k , existiendo aún una brecha con la cota superior conocida, la cual es igual a wk .

1.9 Importancia práctica

Los resultados principales de esta tesis se concentran en la presentación de nuevas cotas inferiores para el problema MTP, mientras que las cotas superiores se mantienen. Esta característica muestra la dificultad de idear algoritmos on-line con competitividades menores a las conocidas previamente.

Los algoritmos on-line para el problema MTP tienen una aplicación práctica en la administración de memoria de sistemas operativos multitarea. En este campo, las nuevas cotas inferiores son en cierta forma desalentadoras, ya que indican que los algoritmos on-line no garantizan una performance suficientemente alta. Esto se debe principalmente a que el modelo representado por el problema MTP no tiene en cuenta la localidad de referencia existente en los datos referenciados por un programa.

Esta dificultad surgió también durante el análisis del problema de Paginación Estándar, donde el factor de competitividad de varios algoritmos on-line (por ejemplo LRU) no se correspondía con la performance observada en la práctica. Para remediar esta discrepancia, Borodin et al. [13] diseñaron un modelo que tiene en cuenta la localidad de referencia. Usando este modelo es posible explicar la performance obtenida por el algoritmo LRU. Este trabajo fue extendido por Irani et al. [16].

Futuras investigaciones en MTP podrían seguir este curso de acción con el objetivo de idear algoritmos on-line buena competitividad y buena performance en la práctica.

2. Restricciones sobre algoritmos e instancias

2.1 Introducción

En este capítulo veremos algunos resultados generales sobre el problema MTP.

Mostraremos que podemos restringir el análisis a cierto tipo de algoritmos, concretamente a algoritmos *lazy* (*perezosos*). Veremos también que podemos restringir aún más el conjunto de algoritmos y analizar sólo aquellos que sirven inmediatamente todos los pedidos a páginas que están en la caché.

Definiremos los conceptos de instancias suaves y duras. Mostraremos que el adversario no tiene motivación para generar instancias suaves, pues todo algoritmo c -competitivo en todas las instancias duras puede convertirse fácilmente en uno c -competitivo para instancias suaves y duras.

Finalmente, mostraremos un resultado válido para cualquier problema on-line que usaremos repetidamente en los capítulos posteriores. Este resultado nos permite obtener cotas inferiores para la competitividad de los algoritmos on-line.

2.2 Algoritmos Lazy

Un algoritmo se llama *lazy* si para servir un pedido a una página que no se encuentra en la caché desaloja una sola página, y para servir un pedido a una página que se encuentra en la caché no desaloja ninguna.

A partir de esta definición vemos el motivo de la denominación *lazy*. Cuando se le presenta un pedido, el algoritmo lo sirve con la menor cantidad de page-faults posibles: ninguno si la página pedida está en la caché y uno si no está.

Los algoritmos que no son *lazy* son de alguna manera irracionales: desalojan páginas innecesariamente. Todo algoritmo con comportamiento no *lazy* se convierte fácilmente en uno *lazy* simplemente evitando los page-faults innecesarios y postergando los necesarios hasta el momento en que se necesite la página cargada.

El siguiente lema muestra que podemos restringir nuestra atención a algoritmos *lazy*. Un resultado similar fue probado por Manasse et al.[9] para el problema de paginación estándar. Extenderemos su resultado para al problema MTP.

Lema 2.1: Para todo algoritmo A para el problema MTP existe un algoritmo A' para el problema MTP que es lazy, no tiene un costo mayor al de A, y es on-line si A lo es.

Demostración:

El orden en que el algoritmo A sirve los pedidos de las múltiples secuencias define una secuencia de pedidos que llamaremos s . Por el resultado de Manasse et al.[9] sabemos que existe un algoritmo lazy B para paginación estándar que sirve esta secuencia de pedidos con un costo que no puede ser mayor al costo del algoritmo A. Esto se debe a que una vez fijada la secuencia s se puede ver al algoritmo A como un algoritmo para paginación estándar. Notar que si el algoritmo A es on-line también lo es el algoritmo B.

Definamos al algoritmo A' como el algoritmo que sirve los pedidos de las múltiples secuencias en el mismo orden que A, y carga y descarga páginas de la caché igual que el algoritmo B sobre la secuencia s .

El algoritmo A' es lazy pues B lo es, es on-line si A lo es, y tiene un costo igual al de B. Como el costo de B es menor o igual al de A, el algoritmo A' tiene un costo menor o igual al de A.

■

Todos los algoritmos presentados en este trabajo serán lazy. Con esto no se pierde generalidad, pues el lema anterior nos asegura que con estos se puede obtener por lo menos la misma performance que con los algoritmos que no son lazy.

2.3 Servicio de Páginas en la Caché

Además de ser lazy, los algoritmos presentados en este trabajo tendrán otra propiedad: servirán inmediatamente los pedidos a páginas que estén en la caché. Estos pedidos son gratis, por lo que el costo del algoritmo no puede ser mayor que el de un algoritmo que los posterga. El siguiente lema demuestra esta afirmación.

Diremos que un algoritmo A para MTP realiza una *postergación* cuando sirve un pedido a una página que no se encuentra en la caché, y existe un pedido posible de servir a una página que se encuentra en la caché.

Lema 2.2: Para todo algoritmo A para el problema MTP existe un algoritmo A' para el problema MTP que no realiza ninguna postergación, no tiene un costo mayor al de A , y es on-line si A lo es.

Demostración:

Si el algoritmo A no es lazy el lema 2.1 nos asegura que existe un algoritmo que es lazy, no tiene un costo mayor al de A , y es on-line si A lo es. Probando el lema para este algoritmo lazy, se estaría probando también para el algoritmo A . Por lo tanto, alcanza con suponer que A es lazy.

Definamos al algoritmo A' como el algoritmo que emula al algoritmo A en cada servicio donde A no realiza una postergación.

Cuando el algoritmo A realiza una postergación, el algoritmo A' sirve en cambio todos los pedidos a páginas que se encuentren en la caché, hasta no poder servir más pedidos gratuitamente.

Cuando el algoritmo A sirve un pedido ya servido por el algoritmo A' , el algoritmo A' simplemente no sirve ninguna página.

Claramente se ve que todo page-fault del algoritmo A' es también un page-fault del algoritmo A , pues si A' sirve un pedido a una página que no se encuentra en la caché, el algoritmo A también sirve ese pedido sin que la página pedida se encuentre en la caché. Por lo tanto, el costo del algoritmo A' no puede ser mayor que el costo del algoritmo A . ■

En consecuencia, si un algoritmo on-line recibe un pedido a una página que se encuentra en la caché puede servirlo inmediatamente sin temor a estar incrementando su costo innecesariamente. Esto sugiere que el adversario no tiene incentivo para generar instancias con pedidos a páginas que el algoritmo tiene en la caché, pues éste las serviría gratuitamente.

Parecería que esto significa que el adversario va a generar instancias en las que el algoritmo on-line falle en todos los pedidos, pero no es así. Veamos el siguiente ejemplo para el caso en el cual $k=1$ y $w=2$, donde el algoritmo on-line acaba de servir el pedido a la página 5, y el adversario debe decidir el siguiente pedido de la secuencia S_2 :

S_1	S_2
<i>1</i>	<i>2</i>
3	<i>5</i>
	?

En la tabla anterior sólo se muestran los segmentos de las secuencias que el algoritmo on-line conoce. Las secuencias completas pueden ser mayores o no, a decisión del adversario. Las páginas de pedidos ya servidos se muestran en negrita y cursiva.

Como el algoritmo on-line acaba de servir el pedido a la página 5, en la caché se encuentra la página 5. El lema anterior nos dice que para el adversario sería inútil agregar a S_2 un pedido a la página 5, pues el algoritmo podría servirlo inmediatamente en forma gratuita. Sin embargo, el adversario podría decidir agregar a S_2 un pedido a la página 3, pues esta página no se encuentra en la caché. El algoritmo on-line posiblemente servirá un pedido a la páginas 3 y luego el otro, sirviendo uno de ellos gratis.

S_1	S_2
1	2
3	5
	3

El lema 2.2 no nos dice nada acerca de esta configuración. El adversario podría tener diversos motivos para pedir la página 3. Por ejemplo, podría desear realizar un nuevo pedido a la página 5 en la secuencia S_2 , pero si los pone consecutivos el algoritmo on-line podría servirlos juntos. Entonces intercala un pedido a la página 3.

Por lo tanto podemos suponer que el adversario no tiene incentivo para hacer un pedido a una página que se encuentra en la caché, pero no podemos suponer que el adversario no tiene incentivo para generar secuencias donde se sirvan pedidos gratuitamente.

2.4 Instancias Suaves y Duras

Llamaremos *suave* a una instancia que presente un par de pedidos consecutivos iguales en alguna de sus secuencias. En caso contrario la llamaremos *dura*.

El lema 2.2 nos permite inferir que si un algoritmo sirve los pedidos consecutivos iguales juntos, como una unidad, no puede incurrir en un costo mayor. Por lo tanto, el adversario no tiene motivación alguna para generar instancias suaves, pues todo algoritmo c -competitivo en todas las instancias duras puede convertirse fácilmente en uno c -competitivo para instancias suaves y duras.

Lema 2.3: Para todo algoritmo A para el problema MTP existe un algoritmo A' para el problema MTP que sirve los pedidos consecutivos iguales juntos, no tiene un costo mayor al de A , y es on-line si A lo es.

Demostración:

Cuando el algoritmo A ve por primera vez un pedido igual al anterior en la misma secuencia, la página pedida debe estar necesariamente en la caché. Por lo tanto, el mismo algoritmo A' obtenido en el lema 2.2 cumple las condiciones del lema, con la

ligera modificación de que si hay pedidos consecutivos iguales, A' los sirve juntos. La modificación es necesaria pues el algoritmo A' podría separarlos sirviendo entre ellos pedidos a otras páginas que también se encuentren en la caché.

El algoritmo A' no tiene un costo mayor al de A, y es on-line si A lo es, por lo que cumple las condiciones del lema.

■

Los resultados vistos en este capítulo nos permiten centrar la atención únicamente en algoritmos lazy que sirvan inmediatamente los pedidos gratis. Podemos diseñar los algoritmos para tratar solamente casos en que las instancias son duras, pues pueden convertirse fácilmente para tratar el caso general. Cualquier cota encontrada basándose en estas restricciones es válida para cualquier algoritmo on-line y para cualquier instancia generada por el adversario.

2.5 Instancias con costo no acotado

El resultado que veremos a continuación es válido para cualquier problema, no solamente para MTP. Nos permite obtener cotas inferiores para la competitividad de los algoritmos on-line.

Lema 2.4: Dado un algoritmo A on-line, si para todo entero m existe una instancia I_m tal que $C_o(I_m) > m$ y $C_A(I_m)/C_o(I_m) \geq c'$, entonces el algoritmo A no es c -competitivo con $c < c'$.

Demostración:

Sea A un algoritmo c -competitivo. Por definición de competitividad:

$\forall$ instancia I: $C_A(I) \leq c C_o(I) + b$, para alguna constante $b \Rightarrow$

$\Rightarrow \forall I: C_A(I) / C_o(I) \leq c + b / C_o(I) \Rightarrow$

$\Rightarrow \forall I: C_A(I) / C_o(I) - b / C_o(I) \leq c$

Supongamos que para todo entero m existe una instancia I_m tal que $C_o(I_m) > m$ y $C_A(I_m)/C_o(I_m) \geq c'$. Si $c < c' \Rightarrow c = c' - \varepsilon$ para algún $\varepsilon > 0$.

Sea $m > b/\varepsilon$. Entonces:

$$m > b/\varepsilon \Rightarrow b/C_o(I_m) < b/m < b/(b/\varepsilon) \Rightarrow$$

$$\Rightarrow b/C_o(I_m) < \varepsilon \Rightarrow$$

$$\Rightarrow -b/C_o(I_m) > -\varepsilon \Rightarrow$$

(sumando $C_A(I_m)/C_o(I_m)$ en ambos lados de la desigualdad)

$$\Rightarrow C_A(I_m)/C_o(I_m) - b/C_o(I_m) > C_A(I_m)/C_o(I_m) - \varepsilon \Rightarrow$$

(usando $c \geq C_A(I) / C_o(I) - b / C_o(I)$)

$$\Rightarrow c > C_A(I_m)/C_o(I_m) - \varepsilon \Rightarrow$$

(usando $C_A(I_m)/C_o(I_m) \geq c'$)

$$\Rightarrow c > c' - \epsilon.$$

Esto es absurdo pues $c = c' - \epsilon$. El absurdo provino de suponer $c < c'$.

Por lo tanto, el algoritmo A no es c -competitivo con $c < c'$, probando el lema.

■

Este resultado se puede usar para probar cotas inferiores para la competitividad de los algoritmos on-line. Mostrando que para todo algoritmo on-line el adversario puede generar una instancia tal que:

- el costo óptimo off-line sobre la instancia sea arbitrariamente grande y
- la relación entre los costos del algoritmo on-line y el óptimo off-line sea superior a un número c'

entonces el lema 2.4 nos garantiza que no puede existir un algoritmo c -competitivo con $c < c'$. De esta manera, c' es una cota inferior para la competitividad de los algoritmos on-line. Este procedimiento para encontrar cotas inferiores será usado en los capítulos siguientes.

3. Caché de tamaño uno

3.1 Introducción

En este capítulo comenzamos con el análisis del caso particular del problema MTP en el cual la caché es de tamaño uno.

Veremos que el problema off-line asociado a este caso particular es el conocido problema denominado “Longest Common Subsequence Problem” (LCS - Subsecuencia Común Máxima). Como se vio en el capítulo 1, la solución a este problema es usada en biología molecular, comparación de archivos, etc.

Finalmente, veremos que todo algoritmo on-line lazy para este problema es w -competitivo.

3.2 Un algoritmo off-line óptimo

3.2.1 Dos Secuencias

Una subsecuencia de una cadena s es una cadena que se puede obtener a partir de s eliminando elementos. El problema LCS consiste en hallar la mayor subsecuencia común de dos cadenas.

Si suponemos que las cadenas son secuencias de pedidos a páginas y consideramos una subsecuencia común de estas cadenas como los pedidos que deben ser servidos conjuntamente, vemos que la mejor solución para el problema LCS está relacionada con la solución óptima al problema MTP(1,2) off-line. Llamando S_1 y S_2 a las secuencias y T a una subsecuencia común, el costo de servir los pedidos de S_1 y S_2 sirviendo juntos los pedidos en T es $|S_1| + |S_2| - |T|$, donde $|S|$ es la longitud de la secuencia S . Si T es de máxima longitud este costo es mínimo.

Presentaremos un algoritmo para hallar el mínimo costo de servir una instancia I en el problema MTP(1,2). Este algoritmo es una adaptación directa de un algoritmo para resolver el problema LCS.

El algoritmo arma una matriz L de tamaño $(|S_1|+1) \times (|S_2|+1)$. Cada coeficiente $L[i,j]$ contiene el mínimo costo de servir las subsecuencias $S_1[1...i]$ y $S_2[1...j]$. Por lo tanto, el coeficiente $L[|S_1|,|S_2|]$ contiene el mínimo costo de servir completamente ambas secuencias. Los coeficientes $L[i,0]$ se inicializan en i , los coeficientes $L[0,j]$ se inicializan en j , y el resto de los coeficientes $L[i,j]$ se calculan a partir de los 3 coeficientes $L[i-1,j-1]$, $L[i-1,j]$ y $L[i,j-1]$.

```

FUNCTION CostoMTP(Secuencias  $S_1, S_2$ )
Begin
   $L := \text{Integer}[0..|S_1|, 0..|S_2|]$ 
  For  $i := 0$  to  $|S_1|$ 
     $L[i, 0] := i$ 
  End For

  For  $j := 0$  to  $|S_2|$ 
     $L[0, j] := j$ 
  End For

  For  $i := 1$  to  $|S_1|$ 
    For  $j := 1$  to  $|S_2|$ 
      If  $S_1[i] = S_2[j]$  Then
         $L[i, j] := L[i-1, j-1] + 1$ 
      Else
         $L[i, j] := \min(L[i-1, j] + 1, L[i, j-1] + 1)$ 
      End If
    End For
  End For

  CostoMTP :=  $L[|S_1|, |S_2|]$ 
End.

```

Claramente este algoritmo utiliza programación dinámica y tiene complejidad espacial y temporal $O(|S_1| \cdot |S_2|)$. En 1973 Hirschberg mostró algoritmos más complicados con complejidad espacial $O(\max(|S_1|, |S_2|))$.

El siguiente algoritmo nos permite determinar el orden en que deben ser servidos los pedidos para obtener el costo mínimo a partir de la matriz L generada por la función *CostoMTP*. Recorre la matriz desde la posición $L[|S_1|, |S_2|]$ hasta la posición $L[0,0]$ e indica *en orden inverso* qué secuencias servir en cada paso.

```

FUNCTION Servir(Secuencias  $S_1, S_2$ ,  $L$ :Matriz)
Begin
   $i := |S_1|$ 
   $j := |S_2|$ 
  While  $i > 0$  or  $j > 0$ 

```

```

    If i = 0
        Write('Servir de S2')
        j:=j-1
        Continue
    End If

    If j = 0
        Write('Servir de S1')
        i:=i-1
        Continue
    End If

    If S1[i] = S2[j] and L[i,j] = L[i-1,j-1] + 1 Then
        Write('Servir de S1 y de S2')
        i:=i-1
        j:=j-1
    Else
        If L[i-1,j] < L[i,j-1]
            Write('Servir de S1')
            i:=i-1
        Else
            Write('Servir de S2')
            j:=j-1
        End If
    End If
End While
End.

```

3.2.2 Cualquier número de secuencias

*No demuestra que MSA
resuelve MTP(1,w) por $w > 2$*

El problema LCS se puede generalizar para el caso en que hay más de 2 cadenas. Este problema se denomina "Multiple Sequence Alignment Problem" (MSA – Alineación de Múltiples Secuencias).

Presentaremos una adaptación del algoritmo que resuelve el problema MSA para el problema MTP(1,w). Este algoritmo arma, como en el caso de dos secuencias, una matriz L, pero en este caso de w dimensiones y tamaño $(|S_1|+1) \times (|S_2|+1) \times \dots \times (|S_w|+1)$.

Cada coeficiente $L[i_1, i_2, \dots, i_w]$ se calcula a partir de los coeficientes $L[i_1-v_1, i_2-v_2, \dots, i_w-v_w]$, donde v es un vector de tamaño w y contiene solamente ceros y unos, excluyendo al vector nulo. Por lo tanto hay $2^w - 1$ valores posibles para el vector v. Notar que con dos secuencias tenemos 3 vectores posibles, coincidiendo con el resultado presentado en la sección anterior, donde $L[i,j]$ se calculaba a partir de los 3 coeficientes $L[i-1,j-1]$, $L[i-1,j]$ y $L[i,j-1]$.

Usaremos la notación $L[i]$, donde i es un vector de tamaño w , para referirnos al coeficiente $L[i_1, i_2, \dots, i_w]$.

Las sentencias resaltadas en el pseudo-código siguiente son útiles para la función *Servir* y serán explicadas posteriormente.

```

FUNCTION CostoMTP(Secuencias  $S_1, S_2, \dots, S_w$ )
Begin
     $L := \text{Integer}[0..|S_1|, 0..|S_2|, \dots, 0..|S_w|]$ 
     $M := \text{Integer}[0..|S_1|, 0..|S_2|, \dots, 0..|S_w|]$ 

     $i := \text{Integer}[1..w]$ 
     $v := \text{Integer}[1..w]$ 
     $v' := \text{Integer}[1..w]$ 

     $L[0, 0, 0, \dots, 0] = 0$ 

    For  $i_1 := 0$  to  $|S_1|$ 
    For  $i_2 := 0$  to  $|S_2|$ 
    .
    .
    .
    For  $i_w := 0$  to  $|S_w|$ 
        If  $i = \text{vector nulo}$  Then
            Continue
        End if

        Minimo := infinito.

        ForEach vector  $v$  tal que sus componentes sean 0 y
        1 pero no todas 0, y que  $i-v$  no tenga componentes
        negativas

             $A := \text{cantidad de páginas distintas en el}$ 
             $\text{conjunto de pedidos } \{S_j[i_j]\}, \text{ para los}$ 
             $\text{valores } j \text{ tal que } v_j = 1.$ 

            If  $A = 1$  and  $L[i-v] + 1 < \text{minimo}$  Then
                Minimo :=  $L[i-v] + 1$ 
                 $v' := v$ 
            End IF
        End For

         $L[i] := \text{minimo}$ 
         $M[i] := v'$ 
    End For
    .

```



```

      .
      .
    End For
  End For

  CostoMTP:=L[|S1|, |S2|, ... , |Sw|]
End.

```

Este algoritmo tiene complejidad espacial igual a $O(|S_1| \cdot |S_2| \dots |S_w|)$ y temporal igual a $O(2^w |S_1| \cdot |S_2| \dots |S_w|)$. Se sabe que este problema es NP-Completo con respecto a w (Wang-Jiang [19]).

Un algoritmo análogo al algoritmo *Servir* presentado en la sección anterior nos permite determinar a partir de L el orden en que deben ser servidos los pedidos para obtener el costo mínimo.

Sin embargo, una manera más simple es guardar para cada coeficiente $L[i]$ el vector v que dio origen a su valor. Las sentencias resaltadas en el algoritmo anterior guardan en la matriz M el vector v que resultó en el valor mínimo para cada coeficiente de la matriz L .

El siguiente algoritmo nos permite determinar el orden en que deben ser servidos los pedidos para obtener el costo mínimo a partir de la matriz M generada por la función *CostoMTP*. Recorre la matriz desde la posición $M[|S_1|, |S_2|, \dots, |S_w|]$ hasta la posición $M[0, 0, \dots, 0]$ e indica *en orden inverso* qué secuencias servir en cada paso.

```

FUNCTION Servir(M:Matriz)
Begin
  Armar un vector i[1...w]
  i := {|S1|, |S2|, ... , |Sw|}
  While (i no es el vector nulo)
    For Each componente j del vector M[i] igual a 1
      Write('Servir de Sj')
    End For
    i := i - M[i]
  End While
End.

```

3.3 Algoritmos w -competitivos

En esta sección veremos que todo algoritmo on-line lazy es w -competitivo. Este resultado surge de considerar el caso en que el algoritmo on-line no realiza ningún apareo mientras que el óptimo off-line aparee todos los pedidos. Por lo tanto, es suficiente que un algoritmo on-line no incurra en costos innecesarios (por ejemplo, cargando páginas en la caché que no serán usadas) para que sea w -competitivo.

Lema 3.1: Todo algoritmo on-line lazy para el problema $MTP(1,w)$ es w -competitivo.

Demostración:

Dada una instancia dura I , llamemos l a la longitud de la secuencia de mayor longitud en I .

Sea A un algoritmo on-line lazy para el problema $MTP(1,w)$. Como A es lazy, en cada page-fault sirve por lo menos una página. Como la instancia I tiene a lo sumo wl pedidos en total, el costo $C_A(I)$ del algoritmo A sobre la instancia I no puede ser mayor a wl . Por lo tanto:

$$C_A(I) \leq wl.$$

El costo óptimo off-line de servir la instancia I no puede ser menor al costo de servir una de sus secuencias. Como el costo de servir la secuencia de longitud l no puede ser menor a l pues la instancia I es dura, el costo off-line óptimo $C_o(I)$ de servir la instancia I completa no puede ser menor a l . Por lo tanto :

$$l \leq C_o(I)$$

Consecuentemente $C_A(I) \leq wl \leq w C_o(I)$, para cualquier instancia dura I , por lo que el algoritmo on-line A es w -competitivo sobre las instancias duras.

Si la instancia I no es dura, obtenemos el mismo resultando si llamamos l a la longitud de la mayor secuencia contando los pedidos consecutivos a la misma página como uno solo. Siendo el algoritmo A lazy, sirve los pedidos consecutivos iguales juntos, por lo que su costo no puede ser mayor a wl . El costo óptimo off-line no puede ser menor a l , obteniendo nuevamente $C_A(I) \leq wl \leq w C_o(I)$.

■

4. Caché de tamaño uno y dos secuencias.

4.1 Introducción

En este capítulo analizaremos el problema $MTP(1,2)$, en el cual la caché es de tamaño uno y hay dos secuencias.

En el trabajo realizado por Feuerstein[1] y Feuerstein-Strejilevich de Loma[2] se demostró que la competitividad de ningún algoritmo on-line para el problema $MTP(k,w)$ puede ser menor que $k+1-1/w$. En particular, cuando $k=1$ y $w=2$ esta cota se convierte en $k+1-1/w = 1+1-1/2 = 3/2 = 1.5$.

Los métodos descritos en capítulos siguientes que mejoran las cotas para el problema MTP en general no lo hacen para este caso en particular. Por lo tanto decidimos atacarlo en forma independiente. Con métodos específicos para este caso mostraremos formas de mejorar esta cota inferior. La mejor cota inferior que mostraremos es 1.788 y se obtendrá mediante la utilización de un programa que analiza un árbol de posibilidades. La cota quizás podría mejorarse si se permitiera ejecutar durante más tiempo al programa. Sin embargo, a medida que se intenta encontrar cotas mayores el programa requiere tiempos de cálculo cada vez mayores, y aún para mejoras muy pequeñas el tiempo de cálculo adicional resulta prohibitivo.

En los trabajos realizados por Feuerstein[1] y Feuerstein-Strejilevich de Loma[2] también se mostró una cota superior para el problema $MTP(k,w)$. Esta cota es igual a wk . En el caso particular analizado en este capítulo esta cota se convierte en $wk = 2*1 = 2$. Como vimos en el capítulo anterior, todo algoritmo lazy es 2-competitivo, por lo que encontrar algoritmos que alcancen esta cota superior es relativamente simple. Intentamos encontrar algoritmos c -competitivos con c menor que 2, lamentablemente sin éxito. Sin embargo, mostraremos algunas familias de algoritmos que a priori parecieron prometedores, pero finalmente probaremos que su competitividad no es menor que 2.

También veremos restricciones en las instancias que puede generar el adversario si quiere obtener una relación entre los costos on-line y off-line cercana a 2.

4.2 Relación entre las longitudes de ambas secuencias

Supongamos que en una instancia I las dos secuencias tienen longitudes muy diferentes. En general el algoritmo off-line óptimo no podrá aparear muchos pedidos. Por lo tanto, cuanto más diferentes sean las longitudes, más lejana de 2 será la relación de costos entre el algoritmo on-line y el algoritmo off-line óptimo. Para que el adversario pueda asegurarse una relación de costos alta, debe generar instancias con secuencias de longitud similar.

Sea I una instancia del problema. Llamemos S_1, S_2 a las secuencias de la instancia I y L_1, L_2 a sus longitudes. Sin pérdida de generalidad, podemos suponer que $L_1 \geq L_2$. Llamemos C a C_A / C_0 .

Como consideramos que el adversario genera únicamente instancias con secuencias duras (ver cap. 2), una secuencia no puede tener dos pedidos consecutivos iguales. Por lo tanto el adversario siempre tendrá que pagar por lo menos por la secuencia más larga de la instancia, y su costo será mayor o igual a $\max(L_1, L_2) = L_1$. De esta manera:

$$C_0 \geq L_1$$

Como también consideramos sólo algoritmos on-line lazy, el costo del algoritmo A no puede ser mayor a la cantidad total de pedidos en la instancia I . Por lo tanto:

$$C_A \leq L_1 + L_2$$

Dividiendo los costos de ambos algoritmos obtenemos:

$$C = \frac{C_A}{C_0} \leq \frac{L_1 + L_2}{L_1} = 1 + \frac{L_2}{L_1}$$

$$\Rightarrow C - 1 \leq \frac{L_2}{L_1}$$

$$\Rightarrow L_1 (C - 1) \leq L_2$$

Como $L_1 \leq L_2$, tenemos un rango para L_2 :

$$L_1 (C - 1) \leq L_2 \leq L_1$$

Como A es lazy, C es menor o igual a 2, por lo que $C - 1$ no puede ser mayor que 1. A medida que C se acerca a 2 el rango para L_2 se hace más chico, haciendo que L_1 y L_2 sean muy similares.

Como ejemplo, si el adversario quiere obtener $C = 1.9$ tenemos que:

$$0.9 L_1 \leq L_2 \leq L_1$$

Por lo tanto, las longitudes tienden a ser similares.

Desde el punto de vista del algoritmo on-line, esto significa que debemos buscar algoritmos que obtengan un valor de $C \leq 1.9$ cuando las longitudes son parecidas (la más chica un 90% de la mayor). Cuando las longitudes de las secuencias no cumplan esta

condición no importa lo que haga el algoritmo on-line, pues el valor de C no puede ser mayor a 1.9.

4.3 Máxima cantidad de pedidos no apareados

Podemos realizar un análisis similar al anterior teniendo en cuenta la cantidad de pedidos no apareados por el algoritmo óptimo off-line. Si esta cantidad es grande, el cociente entre el costo del algoritmo on-line y el costo del algoritmo off-line no puede ser muy cercano a 2. Para obtener cotas inferiores cercanas a 2 debemos encontrar instancias donde la proporción de pedidos no apareados por el algoritmo off-line sea pequeña. Cuanto mayor es esta proporción, menor es la relación de costos que se obtiene.

Como en la sección anterior, sea I una instancia del problema. Llamemos S_1, S_2 a las secuencias de la instancia I y L_1, L_2 a sus longitudes. Sin pérdida de generalidad, podemos suponer que $L_1 \geq L_2$. Llamemos C a C_A / C_o .

Llamemos también r_o a la proporción de pedidos que el algoritmo off-line óptimo no aparea de la secuencia S_2 . O sea, r_o es la cantidad de pedidos no apareados de la secuencia de menor longitud dividido por la longitud de esa secuencia. Similarmente llamemos r_A a la proporción de pedidos que el algoritmo óptimo off-line no aparea de la secuencia S_2 .

El costo del algoritmo óptimo off-line es la longitud de la secuencia S_1 más la cantidad no apareada de la secuencia S_2 , pues el resto de las páginas de S_2 se sirven gratis.

El mismo razonamiento para el algoritmo on-line A nos da los siguientes costos:

$$\begin{aligned} C_o &= L_1 + L_2 * r_o \\ C_A &= L_1 + L_2 * r_A \end{aligned}$$

$$\Rightarrow C = \frac{C_A}{C_o} = \frac{L_1 + L_2 * r_A}{L_1 + L_2 * r_o} = \frac{L_1 + L_2 * r_o - L_2 * r_o + L_2 * r_A}{L_1 + L_2 * r_o} = 1 + \frac{L_2(r_A - r_o)}{L_1 + L_2 * r_o} = 1 + \frac{r_A - r_o}{\frac{L_1}{L_2} + r_o}$$

Como $L_1 \geq L_2 \Rightarrow L_1 / L_2 \geq 1$, y obtenemos:

$$\Rightarrow C = 1 + \frac{r_A - r_o}{\frac{L_1}{L_2} + r_o} \leq 1 + \frac{r_A - r_o}{1 + r_o} = \frac{[1 + r_o] + [r_A - r_o]}{1 + r_o} = \frac{1 + r_A}{1 + r_o}.$$

$$\Rightarrow C \leq \frac{1 + r_A}{1 + r_o}$$

Suponiendo el mejor caso para el adversario, el algoritmo on-line no apareea nada, por lo que $r_A = 1$. En general $r_A \leq 1$ y obtenemos:

$$\Rightarrow C \leq \frac{1 + r_A}{1 + r_0} \leq \frac{1 + 1}{1 + r_0} = \frac{2}{1 + r_0}$$

$$\Rightarrow C \leq \frac{2}{1 + r_0}$$

Despejando r_0 de esta ecuación obtenemos:

$$C \leq \frac{2}{1 + r_0} \Rightarrow C * (1 + r_0) \leq 2 \Rightarrow r_0 \leq 2/C - 1$$

Como ejemplo, calculemos el máximo valor de r_0 tal que $C \geq 1.9$.

$$\Rightarrow r_0 \leq 2/C - 1 \leq 2/1.9 - 1 = 1/19$$

Esto significa que la proporción de pedidos no apareados por el algoritmo off-line óptimo de la columna L_2 debe ser menor o igual que $1/19$, o lo que es igual, debe aparearse por lo menos 18 de cada 19 pedidos. Si no fuera así, C no podría ser mayor o igual a 1.9. Si el algoritmo on-line apareea algunos pedidos de la secuencia S_2 , (o sea, $r_A < 1$) el óptimo off-line tendría que tener r_0 todavía más chico. De esta manera se ve que es necesario encontrar un algoritmo on-line que funcione bien cuando las secuencias S_1 y S_2 tienen una gran proporción de pedidos iguales, siendo posible para el adversario aparearlos.

Juntando los resultados de las últimas dos secciones, se ve que si un algoritmo on-line tiene una relación de costos baja con instancias donde las secuencias tienen una gran proporción de pedidos iguales y tienen longitudes parecidas, entonces no importa su funcionamiento en el resto de los casos pues estos resultados aseguran que la competitividad del algoritmo on-line será baja.

4.4 Algoritmos 2-Competitivos

Como se vio en el capítulo 2, todo algoritmo on-line lazy para el problema MTP(1,2) es 2-competitivo. Surge la pregunta de si existe algún algoritmo on-line que tenga una competitividad menor que 2. No hemos podido encontrar ninguno, pero sí pudimos probar que algunos algoritmos *no* tienen una competitividad menor que 2.

Llamemos I a la instancia que el adversario presenta al algoritmo on-line, S_1, S_2 a las 2 secuencias de la instancia I y L_1, L_2 sus longitudes.

Algunos algoritmos lazy que consideraremos son los siguientes:

Mientras no se logre aparear:

- Servir un pedido de S_1 , luego 1 de S_2 , luego 1 de S_1 , luego 1 de S_2 , ...
- Servir un pedido de S_1 , luego 2 de S_2 , luego 3 de S_1 , luego 4 de S_2 , ...
- Servir un pedido de S_1 , luego 2 de S_2 , luego 4 de S_1 , luego 8 de S_2 , ...
- Cualquiera que siempre tenga más pedidos servidos de S_1 que de S_2 .

No importa qué hagan estos algoritmos cuando aparean, pues las instancias que analizaremos no permiten que estos algoritmos apareen ningún pedido.

Todos estos algoritmos son lazy, por lo tanto sabemos que tienen una competitividad menor o igual a 2. Probaremos que ninguno tiene una competitividad menor a 2.

Lema 4.1: Sea A un algoritmo on-line lazy que para toda instancia I , en todo momento ha servido más o igual páginas de la secuencia S_1 que de S_2 . Entonces el algoritmo A no es c -competitivo con $c < 2$.

Evidentemente, el papel de S_1 y S_2 puede ser invertido.

Demostración:

Sea S_2 una secuencia de n páginas distintas. Sea S_1 la secuencia S_2 menos la primera página. Llamemos I_n a la instancia formada por S_1 y S_2 .

Veamos por ejemplo la instancia I_6 :

S_1	S_2
2	1
3	2
4	3
5	4
6	5
-	6
	-

Los guiones al final de las secuencias indican la terminación de las mismas.

Esta instancia tiene costo $2n - 1$ para el algoritmo on-line pues al haber servido siempre más elementos de S_1 que de S_2 nunca puede aparear dos páginas para servir las juntas:

$$C_A(I_n) = 2n - 1.$$

El algoritmo óptimo off-line puede servir S_2 y al mismo tiempo S_1 gratuitamente:

$$C_o(I_n) = n.$$

Supongamos que el algoritmo A es c -competitivo con $c < 2$.

Por lo tanto, $\exists b / \forall \text{ instancia } I \ C_A(I) \leq c \ C_o(I) + b$.

Elijamos n tal que $n > \frac{b+1}{2-c}$. Sobre la instancia I_n tenemos:

$$\begin{aligned} C_A(I_n) - (c C_0(I_n) + b) &= 2n - 1 - cn - b = \\ &= (2-c)n - 1 - b > (2-c) \frac{b+1}{2-c} - 1 - b = 0 \end{aligned}$$

Por lo tanto $C_A(I_n) > c C_0(I_n) + b$, y el algoritmo no es c -competitivo.

El absurdo proviene de suponer que el algoritmo A es c -competitivo con $c < 2$

■

El lema anterior prueba que el primero de nuestros algoritmos (servir un pedido de S_1 , luego 1 de S_2 , luego 1 de S_1 , luego 1 de S_2 , ...) no es c -competitivo con $c < 2$.

Lema 4.2: Sea A un algoritmo on-line lazy que para toda instancia I , nunca sirve más de p pedidos de S_2 que de S_1 . Entonces el algoritmo A no es c -competitivo con $c < 2$.

Demostración:

Sea S_2 una secuencia de n páginas distintas. Sea S_1 la secuencia S_2 menos las primeras $p+1$ páginas. Llamemos $I_{n,p}$ a la instancia formada por S_1 y S_2 .

Veamos por ejemplo la instancia $I_{7,2}$:

S_1	S_2
4	1
5	2
6	3
7	4
-	5
	6
	7
	-

Esta instancia tiene costo $2n - p - 1$ para el algoritmo on-line pues nunca puede aparear dos páginas para servir las juntas:

$$C_A(I_{n,p}) = 2n - p - 1.$$

El algoritmo óptimo off-line puede servir S_2 y al mismo tiempo S_1 gratuitamente:

$$C_0(I_{n,p}) = n.$$

Elijamos n tal que $n > \frac{b+p+1}{2-c}$. Sobre la instancia $I_{n,p}$ tenemos:

$$\begin{aligned}
C_A(I_{n,p}) - (c C_0(I_{n,p}) + b) &= 2n - p - 1 - cn - b = \\
&= (2-c)n - p - 1 - b > (2-c) \frac{b+p+1}{2-c} - p - 1 - b = 0
\end{aligned}$$

Por lo tanto $C_A(I_{n,p}) > c C_0(I_{n,p}) + b$, y el algoritmo no es c -competitivo.

El absurdo proviene de suponer que el algoritmo A es c -competitivo con $c < 2$

■

Los lemas anteriores nos indican que el algoritmo on-line debe de alguna manera avanzar sobre la secuencia S_1 , luego avanzar por la secuencia S_2 , etc. Con esta idea, separaremos las instancias en fases. Una fase termina cuando se han servido la misma cantidad de pedidos de ambas secuencias o se termina de servir la instancia. Por ejemplo, si en un momento se sirvieron 6 páginas de S_1 y 3 de S_2 , la fase termina cuando S_2 'alcance' a S_1 , o sea cuando el algoritmo A haya servido tantos pedidos de S_2 como de S_1 .

Definamos para cada algoritmo on-line una función $f(I, n)$ siendo I la instancia y n el número de fase. $f(I, n)$ será la cantidad de pedidos servidos de cada secuencia en la fase n . Como al comenzar una fase se ha servido la misma cantidad de pedidos de cada secuencia, y al terminar también, la cantidad de pedidos servidos de cada secuencia en la fase es igual (salvo en la última fase).

Los algoritmos presentados anteriormente tienen la particularidad de que mientras no encuentren un apareo, su comportamiento no depende de I . Por lo tanto, analicemos cómo puede ser la función f en esos casos y usemos $f(n)$ en lugar de $f(I, n)$. Como la función f no indica cuál de las dos secuencias es la que en cada fase debe alcanzar a la otra, varios algoritmos on-line pueden compartir la misma función f . Esta particularidad no influye en los resultados siguientes.

Llamemos $M_R = \sum_{i=1}^R f(i)$. M_R indica la cantidad de pedidos servidos de cada secuencia al finalizar la fase r .

Lema 4.3: Si $\lim_{n \rightarrow \infty} f(n) = \infty$, entonces $\lim_{R \rightarrow \infty} (R/M_R) = 0$.

Demostración:

Si $\lim_{n \rightarrow \infty} f(n) = \infty$, $\forall j (\exists P_j / f(P_j) > j \forall P' > P_j)$.

$\Rightarrow$ Dado j , $\forall R > P_j \Rightarrow$

$$M_R = \sum_{i=1}^R f(i) = \sum_{i=1}^{P_j} f(i) + \sum_{i=P_j+1}^R f(i) > \sum_{i=1}^{P_j} f(i) + \sum_{i=P_j+1}^R j = \sum_{i=1}^{P_j} f(i) + j(R - P_j).$$

Fijado j , $\sum_{i=1}^{P_j} f(i)$ es una constante con respecto a R , que llamamos $\Phi_j \Rightarrow$

$$\begin{aligned} M_R &> \Phi_j + j(R - P_j) \Rightarrow \\ &\Rightarrow \forall R > P_j, R/M_R < R/[\Phi_j + j(R - P_j)] \Rightarrow \\ &\Rightarrow \forall R > P_j, R/M_R < 1/[\Phi_j/R + j - P_j/R]. \end{aligned}$$

Notando que $\lim_{R \rightarrow \infty} 1/[\Phi_j/R + j - P_j/R] = 1/j$, es fácil ver que $\lim_{R \rightarrow \infty} (R/M_R) = 0$:

$$\lim_{R \rightarrow \infty} (R/M_R) = 0 \Leftrightarrow (\forall \varepsilon > 0 \exists r \text{ tal que } R > r \Rightarrow |R/M_R| < \varepsilon).$$

Como $R/M_R > 0 \forall R$, $|R/M_R| = R/M_R$. Por lo tanto alcanza con ver que $\forall \varepsilon > 0 \exists r$ tal que $R > r \Rightarrow R/M_R < \varepsilon$.

Tomando $j > 1/\varepsilon$, como $\lim_{R \rightarrow \infty} 1/[\Phi_j/R + j - P_j/R] = 1/j < \varepsilon$, tenemos que $\exists r \geq P_j$ tal que $R > r \Rightarrow 1/[\Phi_j/R + j - P_j/R] < \varepsilon \Rightarrow R/M_R < \varepsilon$.

■

Lema 4.4: Dado un algoritmo on-line A , sea $f(n)$ la función de pedidos servidos en cada fase. Si $\lim_{n \rightarrow \infty} f(n) = \infty$, entonces el algoritmo A no es c -competitivo con $c < 2$.

Demostración:

Para un algoritmo de este tipo, el adversario puede dar la siguiente instancia:

Dada una cantidad de fases R , armemos S_1 poniendo un pedido a la página 1 en las posiciones $M_{r-1}+1$ con r impar, un pedido a la página 3 en las posiciones $M_{r-1}+1$ con r par, y entremedio de esta páginas rellenar con la secuencia $[5,6,5,6,\dots]$ (Para realizar esta construcción asumimos que $M_0=0$).

La secuencia S_2 es igual a S_1 pero reemplazando los pedidos a las páginas 1 y 3 por los pedidos a las páginas 2 y 4, respectivamente.

A continuación hay un ejemplo con $f(n)=n$. (Esta función se corresponde con el algoritmo: servir un pedido de S_1 , luego tres de S_2 , luego cinco de S_1 , luego siete de S_2 , ...)

1	2	$f(1)$
3	4	
5	5	$f(2)$
1	2	
5	5	
6	6	$f(3)$
3	4	
5	5	
6	6	
5	5	$f(4)$
1	2	
5	5	
6	6	
5	5	
6	6	$f(5)$
3	4	
5	5	
.....		
5	5	$f(R)$

Sobre la instancia generada, el costo del algoritmo on-line es la suma de las longitudes de las secuencias pues falla en todos los pedidos. Esto es igual a 2 veces el largo de cada una, que es igual a $2M_R$.

El algoritmo óptimo off-line puede servir las secuencias juntas, salvo en las posiciones M_R+1 donde difieren. Hay R de estas posiciones. Por lo tanto su costo sobre la instancia es M_R+R .

La competitividad del algoritmo no puede ser menor al cociente de estos costos:

$$c \geq \frac{2M_R}{(M_R+R)} = 2 \frac{1}{1 + \frac{R}{M_R}}$$

Como $\lim_{n \rightarrow \infty} f(n) = \infty$ por hipótesis, el lema 4.3 nos dice que $\lim_{R \rightarrow \infty} (R/M_R) = 0$. Entonces:

$$c \geq \lim_{R \rightarrow \infty} 2 \frac{1}{1 + \frac{R}{M_R}} = 2 \frac{1}{1+0} = 2.$$

■

Este último lema incluye todos los algoritmos de tipo:

- Servir un pedido de S_1 , luego 2 de S_2 , luego 3 de S_1 , luego 4 de S_2 , ...
- Servir un pedido de S_1 , luego 2 de S_2 , luego 4 de S_1 , luego 8 de S_2 , ...
- etc.

Por lo tanto, ninguno de estos algoritmos es c -competitivo con $c < 2$.

4.5 Nueva cota inferior

En esta sección mostraremos cómo el adversario puede generar instancias para un algoritmo on-line A de modo de asegurarse una competitividad por lo menos de p , siendo p un parámetro para la generación de las instancias. Usando este método mostraremos que una cota inferior para $MTP(1,2)$ es 1.788, aunque podría usarse el método para obtener un valor ligeramente superior.

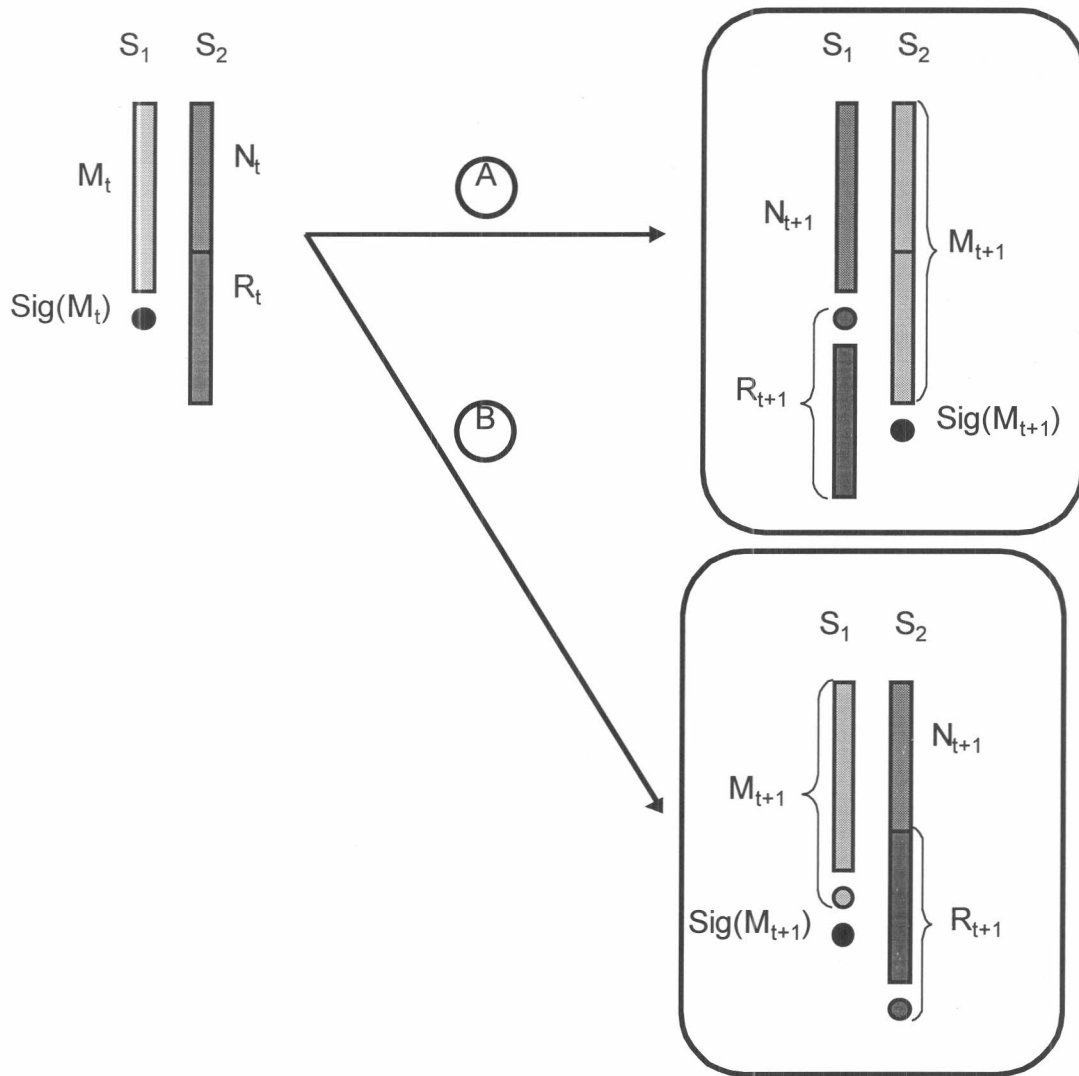
4.5.1 Generación de las instancias

La generación de las instancias depende de un parámetro p : el valor de la cota inferior que el adversario desea obtener. Para ciertos valores de p el adversario será capaz de generar la instancia y para otros no. Por lo tanto, deseamos saber el máximo valor de p para el cual las instancias se pueden generar.

Para valores chicos de p se puede mostrar mediante un grafo pequeño en forma de árbol la manera de generar la instancia I para cada algoritmo on-line. Sin embargo, a medida que p crece, se necesita una computadora para evaluar todas las acciones que puede realizar el algoritmo on-line y armar la instancia I necesaria.

Llamemos S_1, S_2 a las 2 secuencias de la instancia I y L_1, L_2 sus longitudes.

La generación de la instancia se realiza en fases. Estas fases no son las mismas que en la sección anterior, y posteriormente definiremos dónde comienza cada una. En cada fase el adversario mantiene 3 secuencias: M_t , N_t , y R_t , donde t indica el número de fase. Al finalizar cada fase los valores de estas 3 secuencias se actualizan. La idea principal de estas secuencias es la siguiente: M y N son las subsecuencias de S_1 y S_2 que el algoritmo on-line ya sirvió, mientras que R es una secuencia de páginas que el adversario agregará a la instancia I una vez terminada la fase. Formalizaremos este concepto con ayuda del siguiente gráfico:



Supongamos que cuando el adversario comienza una fase los valores de M , N y R son los indicados a la izquierda de la figura. El algoritmo on-line sirvió toda la secuencia M y la N . Está viendo el pedido siguiente a M (representado por un punto negro debajo de M) y el primer pedido de R . El resto de los pedidos de R aún no ha sido visto por el algoritmo on-line. Desde el punto de vista del adversario, al algoritmo on-line puede realizar sólo dos acciones:

- 1) Servir todos los pedidos de R : esta acción la llamaremos acción A
- 2) Servir el pedido siguiente a M : esta acción la llamaremos acción B

Notar en la acción A el término ‘todos’: significa que si el algoritmo on-line sirve algunos pedidos de R pero no todos, y luego sirve la página siguiente a M , para el adversario es como si no hubiese servido ningún pedido de R y sólo realizó la acción B.

En cada fase las secuencias M , N y R se identifican con una de las secuencias S_1 y S_2 , y luego de una acción A cambian y se identifican con la otra. Como ejemplo, en el gráfico anterior, M se identifica con S_1 , y luego de que el algoritmo on-line realiza la acción A, M se identifica con S_2 .

La fase termina precisamente cuando el algoritmo on-line realiza alguna de las dos acciones. Los valores de M , N y R se actualizan dependiendo de la acción tomada por el algoritmo on-line como muestra el gráfico. En gris claro se ve el valor de M , antes y después de la acción del algoritmo on-line. En gris oscuro se ve el valor de N . Con trama rallada se muestra el valor de R . El punto negro es el pedido que sigue a M , y lo denotamos con $\text{Sig}(M)$.

Como es necesario diferenciar los distintos valores de M y N en distintas fases, utilizamos el subíndice t para indicar sus valores en la fase t . Por ejemplo, en la fase 4 los valores de M y N son M_4 y N_4 . También, abusando un poco la notación, utilizamos la misma forma para indicar la secuencia y para indicar su longitud. Por lo tanto M_4 es tanto la secuencia M en la fase 4 como su longitud.

Veamos la respuesta del adversario a cada acción posible del algoritmo on-line:

Acción A:

M_{t+1} es N_t concatenado con R_t

N_{t+1} es M_t

R_{t+1} es el pedido que seguía a M_t concatenado con la secuencia R_t

Si el adversario desea continuar con la construcción de la instancia, agrega un pedido a una página nueva, diferente a todas las aparecidas anteriormente, en la secuencia identificada con R_t

En caso contrario, termina la secuencia identificada con R_t

Informalmente, cuando se realiza la acción A, el adversario agrega a la secuencia identificada con M una copia de los pedidos servidos de R y un pedido nuevo a la otra secuencia. Luego intercambia las secuencias identificadas con N y R con M .

Acción B:

M_{t+1} es M_t concatenado con el pedido que seguía a M_t

N_{t+1} es N_t (o sea, no cambia)

R_{t+1} es R_t concatenado con el pedido que seguía a M_t

Si el adversario desea continuar con la construcción de la instancia, agrega un pedido a una página nueva, diferente a todas las aparecidas anteriormente, en la secuencia identificada con M_t

En caso contrario, termina la secuencia identificada con M_t

Informalmente, cuando se realiza la acción B el adversario agrega el pedido servido al final de la secuencia identificada con R y un pedido a una página nueva a la secuencia identificada con N .

Siempre se puede suponer sin pérdida de generalidad que el algoritmo on-line comienza con la secuencia S_1 . Por lo tanto los valores iniciales para las secuencias M , N y R se pueden obtener del siguiente diagrama (el pedido en negrita es el primero servido):

S_1	S_2
1	2
3	1

Por lo tanto los valores iniciales son:

Sec.	Tamaño	Valor
M_0	1	$\{1\}$
N_0	0	$\{\}$
R_0	2	$\{2,1\}$
$\text{Sig}(M_0)$	1	$\{3\}$

4.5.2 Instancias de costo arbitrariamente grande:

La construcción de las instancias presentada anteriormente tiene secuencias de una longitud determinada para cada algoritmo on-line dado. Por lo tanto, el costo del algoritmo on-line sobre estas instancias está acotado superiormente por una constante b . Esto nos impide demostrar cotas inferiores a la competitividad de un algoritmo on-line.

Sin embargo, es fácil hacer las instancias con secuencias tan largas como se desee reemplazando cada pedido de la construcción anterior por una secuencia de l pedidos a páginas únicas. Cada vez que aparece un pedido a la misma página hay que reemplazarlo por la misma secuencia de l pedidos, y las secuencias que reemplazan pedidos a páginas distintas deben tener pedidos a páginas distintas. Esta modificación no influye en la creación de la instancia, pues el adversario considera cada secuencia de longitud l como una unidad: considera sólo los casos en que se la sirvió completamente o no se la sirvió nada. Sólo cuando se la sirvió completamente el adversario decide los siguientes pedidos de la instancia.

Este cambio multiplica el costo del algoritmo on-line por la constante l . Por otro lado, el costo del algoritmo off-line es menor o igual al anterior multiplicado por l , por lo que la relación entre los costos no disminuye.

4.5.3 Análisis de la instancia generada

La instancia generada de esta manera tiene dos propiedades importantes:

Lema 4.5: El algoritmo on-line falla en todos los pedidos.

Demostración:

La página del pedido que sigue a M es única por construcción. Por lo tanto este pedido no se puede aparear con ninguno de R .

■

Lema 4.6: La secuencia N_t está contenida en M_t , mientras que M_t está contenida en la secuencia $N_t + R_t$.

Demostración:

Dado que $N_0 = \{\}$, $M_0 = \{1\}$ y $N_0 + R_0 = \{2,1\}$, para $t=0$ la propiedad se cumple.

Veamos por inducción que la propiedad se cumple para $t+1$ si se cumple para t . Llamaremos $\text{Sig}(M_t)$ al pedido siguiente a la secuencia M_t .

Acción A:

$$M_{t+1} = N_t + R_t$$

$$N_{t+1} = M_t$$

$$R_{t+1} = \text{Sig}(M_t) + R_t$$

Como $N_{t+1} = M_t \subset N_t + R_t = M_{t+1} \Rightarrow N_{t+1} \subset M_{t+1}$

Como $M_{t+1} = N_t + R_t \subset M_t + R_t \subset M_t + [\text{Sig}(M_t) + R_t] = N_{t+1} + R_{t+1} \Rightarrow M_{t+1} \subset N_{t+1} + R_{t+1}$

Acción B:

$$M_{t+1} = M_t + \text{Sig}(M_t)$$

$$N_{t+1} = N_t$$

$$R_{t+1} = R_t + \text{Sig}(M_t)$$

Como $N_{t+1} = N_t \subset M_t \subset M_t + \text{Sig}(M_t) \subset M_{t+1} \Rightarrow N_{t+1} \subset M_{t+1}$

Como $M_{t+1} = M_t + \text{Sig}(M_t) \subset [N_t + R_t] + \text{Sig}(M_t) = N_t + [R_t + \text{Sig}(M_t)] = N_{t+1} + R_{t+1} \Rightarrow M_{t+1} \subset N_{t+1} + R_{t+1}$.

■

Lema 4.7: Si la instancia se termina de construir en la fase t , el algoritmo on-line no es c -competitivo con $c < 1 + \frac{M_t}{N_t + R_t}$

Demostración:

El lema 4.5 nos dice que el algoritmo on-line falla en todos los pedidos. Como la cantidad de pedidos de una instancia que se terminó de construir en la fase t es $M_t + N_t + R_t$, tenemos que:

$$C_A = M_t + N_t + R_t.$$

El lema 4.6 nos dice que la secuencia M_t está contenida en la secuencia N_t+R_t . Por lo tanto el algoritmo óptimo off-line puede servir la secuencia N_t+R_t , y junto con ella la secuencia M_t , incurriendo en un costo a lo sumo N_t+R_t :

$$C_o \leq N_t+R_t.$$

El cociente entre estos costos es:

$$\frac{C_A}{C_o} \geq \frac{M_t+N_t+R_t}{N_t+R_t} = 1 + \frac{M_t}{N_t+R_t}.$$

Como la instancia generada puede hacerse de costo arbitrariamente grande (ver 4.5.2) y el cociente entre los costos es mayor o igual a $1 + \frac{M_t}{N_t+R_t}$, el lema 2.4 nos asegura que el algoritmo on-line no es c -competitivo con $c < 1 + \frac{M_t}{N_t+R_t}$.

■

Este lema muestra que el adversario puede terminar la construcción de la instancia I cuando detecta que $1 + \frac{M_t}{N_t+R_t} \geq p$ pues se asegura que p es una cota inferior a la competitividad del algoritmo on-line.

A partir de este punto los cálculos resultan engorrosos a menos que hagamos un cambio de variables. Definamos $L_t = N_t + R_t$ y reescribamos las ecuaciones sin N_t ni R_t . De esta manera los cálculos dependen sólo de M , L y t (3 valores) en vez de M , N , R y t (4 valores). Sólo nos interesa la longitud de las secuencias, por lo que a partir de ahora M_t , N_t , R_t y L_t son sólo números.

Lema 4.8: $R_t = t+2$

Demostración:

$$R_0 = 2 = t+2$$

Acción A:

$$R_{t+1} = |\text{Sig}(M_t)| + R_t = 1 + (t+2) = (t+1) + 2.$$

Acción B:

$$R_{t+1} = R_t + |\text{Sig}(M_t)| = (t+2) + 1 = (t+1) + 2.$$

■

Lema 4.9: Si $L_t = N_t + R_t \Rightarrow$ las siguientes ecuaciones son equivalentes a las primeras:

Acción A:

$$\begin{aligned} M_{t+1} &= L_t \\ L_{t+1} &= M_t + t + 3 \end{aligned}$$

Acción B

$$\begin{aligned} M_{t+1} &= M_t + 1 \\ L_{t+1} &= L_t + 1 \end{aligned}$$

Demostración:

Acción A:

$$\begin{aligned} M_{t+1} &= N_t + R_t = L_t \\ L_{t+1} &= N_{t+1} + R_{t+1} = M_t + (t+1) + 2 = M_t + t + 3 \end{aligned}$$

Acción B

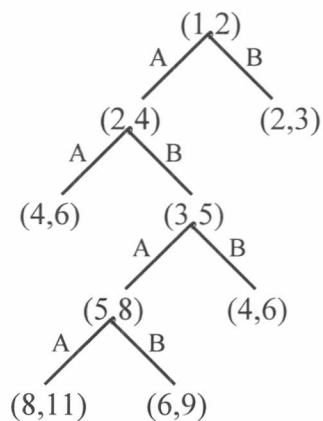
$$\begin{aligned} M_{t+1} &= M_t + |\text{Sig}(M_t)| = M_t + 1 \\ L_{t+1} &= N_{t+1} + R_{t+1} = N_t + (t+1) + 2 = N_t + (t+2) + 1 = N_t + R_t + 1 = L_t + 1 \end{aligned}$$

■

Por lo tanto, hemos eliminado una variable. Utilizando las nuevas variables, el lema 4.7 se convierte en que la competitividad del algoritmo on-line es mayor o igual a $\frac{M_t + L_t}{L_t} = 1 + \frac{M_t}{L_t}$, una expresión ligeramente más simple.

El siguiente gráfico muestra el árbol de posibilidades de acción del algoritmo on-line para $p = 5/3 = 1.666\dots$. En cada nodo se muestra el valor de M_t y de L_t . La altura del nodo es el valor de t . Cuando $\frac{M_t + L_t}{L_t} \geq p$ el árbol deja de generarse, pues el adversario termina la generación de la instancia. Si $\frac{M_t + L_t}{L_t} < p$, hay que extender la rama con las 2 acciones posibles: A y B. En el árbol definimos arbitrariamente que la acción A extiende la rama hacia la izquierda y la acción B la extiende hacia la derecha.

Notar que este gráfico es una prueba de que la competitividad es mayor o igual a $5/3$, mejorando la cota anterior conocida de 1.5 .



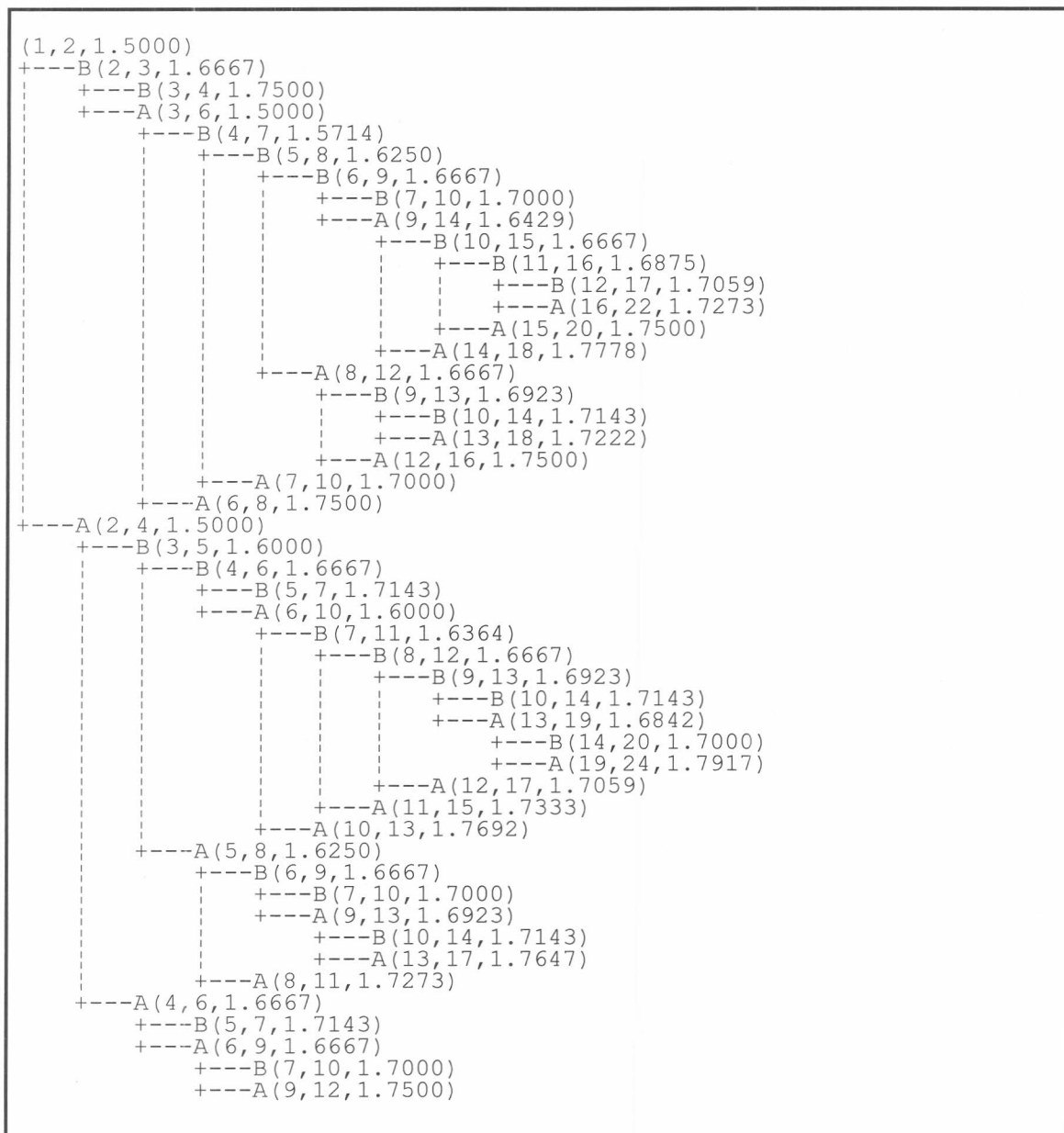
Este ejemplo muestra el método general para demostrar una cota inferior para el problema MTP(1,2). Hay que proponer una competitividad p . Armar el árbol de acciones posibles del algoritmo on-line. Si el árbol no tiene ninguna rama infinita, entonces no existe algoritmo on-line c -competitivo con $c < p$. En cambio, si el árbol tiene una rama infinita no se puede decir nada sobre la competitividad.

4.5.4 Algunos árboles

El siguiente árbol se realizó con $p = 1.70$.

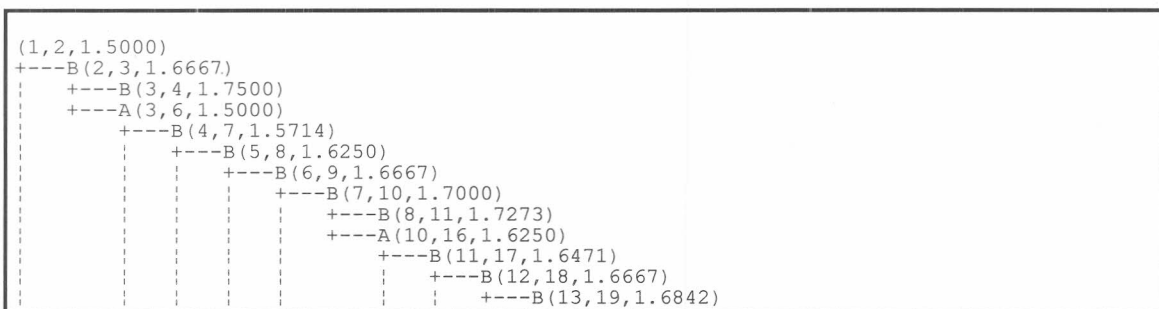
Los datos en el nodo son: La acción del algoritmo on-line(A o B), M_t , L_t y $1+M_t/L_t$, en ese orden.

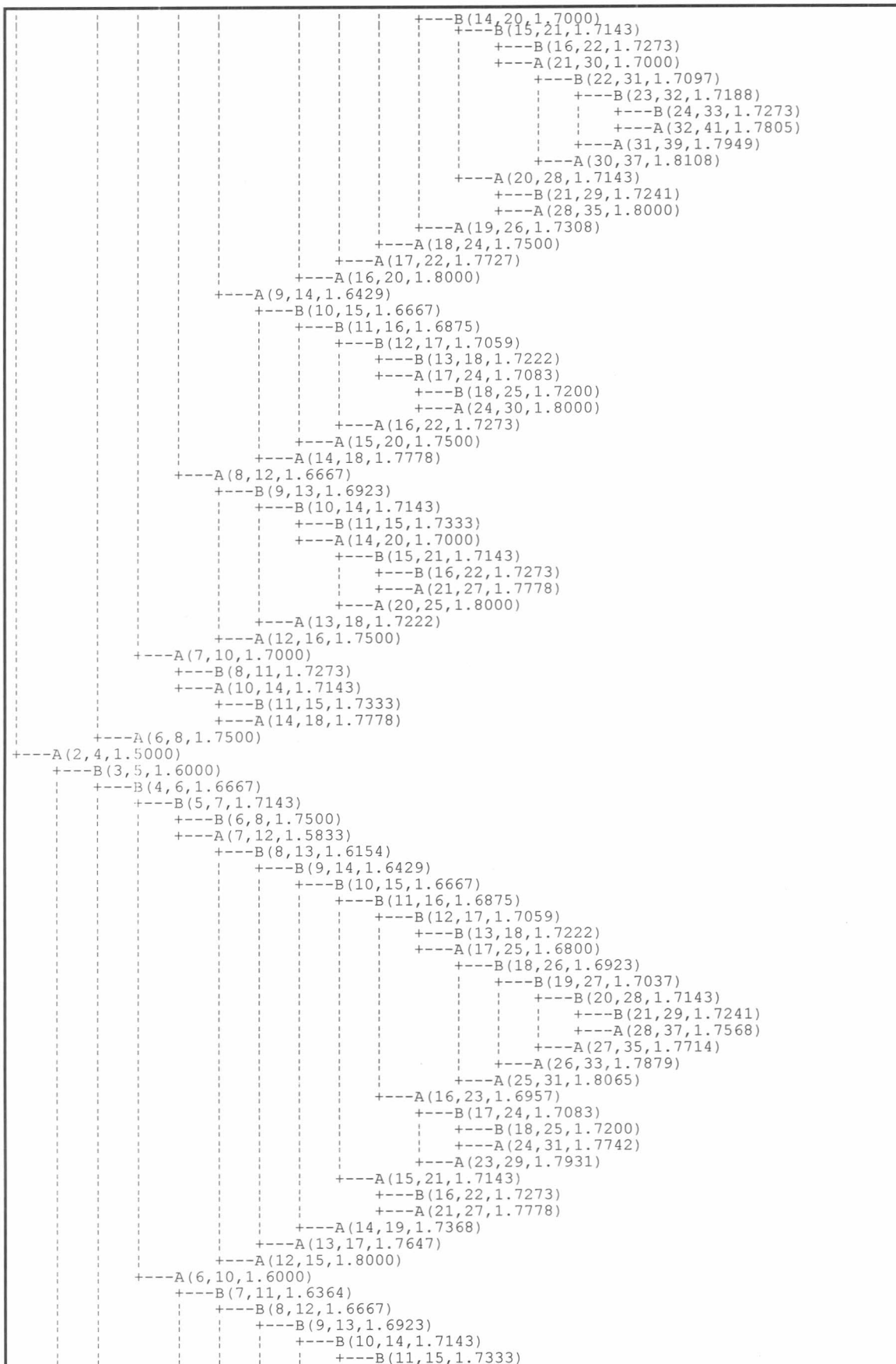
Se puede ver que las hojas tienen un valor de $1+M_t/L_t$ mayor o igual a p . Por lo tanto este árbol es una prueba de que la competitividad de todo algoritmo on-line es mayor o igual a 1.7, mejorando la cota inferior de $5/3$ mostrada en el árbol anterior.

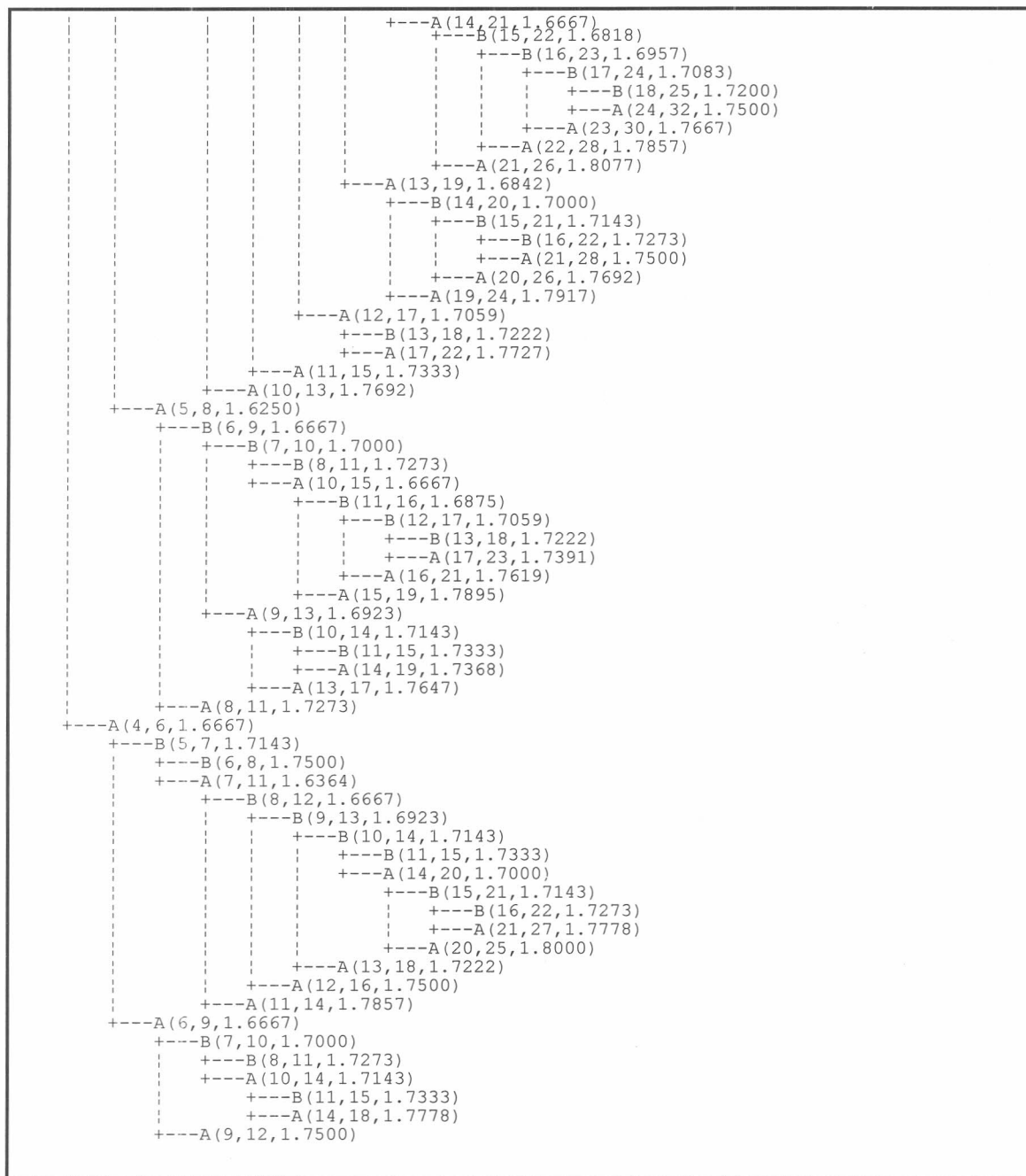


El siguiente árbol se realizó con $p = 1.72$.

Este árbol es una prueba de que la competitividad de todo algoritmo on-line es mayor o igual a 1.72, mejorando la cota de 1.7 mostrada en el árbol anterior.







Se ve claramente el crecimiento del árbol a medida que se agranda el parámetro p :

Con $p=5/3=1.666\dots$ el árbol es relativamente chico y pudo ser hecho a mano.

Con $p=1.70$ el árbol es bastante mayor.

Con $p=1.72$ el árbol es demasiado grande como para poder ser generado sin una computadora.

Por lo tanto, pequeños incrementos en p producen árboles (y tiempo de cálculo) cada vez mayores.

A partir del análisis previo surge naturalmente el siguiente problema:

Hallar el máximo valor para p tal que el árbol tenga todas sus ramas finitas.

El ataque algebraico es el que parece más razonable. Sin embargo, la complejidad de tener 2 tipos de acciones en una ecuación recursiva no nos permitió llegar a ningún resultado concluyente, aunque pudimos aislar varias propiedades de la recursión.

Por lo tanto, decidimos implementar un programa que analice el árbol generado e indique si es un árbol finito. Cuando el árbol generado no es finito, el programa continúa analizando las ramas hasta que finalmente encuentra una rama con una profundidad mayor a la máxima profundidad permitida indicada al programa mediante un parámetro. Aunque esto no demuestra que el árbol es infinito, al variar la máxima profundidad del árbol podemos obtener una 'idea' de si la rama es muy profunda o si es infinita.

4.5.5 Programa para analizar el árbol generado

El programa que analiza el árbol generado a partir de un parámetro p es simple: debe realizar la recursión y parar cuando $1+M_t/L_t \geq p$. O finalizar si la profundidad de la recursión alcanza la mayor profundidad permitida.

El siguiente pseudo-código muestra cómo puede hacerse:

```
Proc Principal
Begin
  M:=1
  L:=2
  T:=0
  Procesar(M,L,T)
  Write('El árbol es finito!')
End.
```

```
Proc Procesar(M,L,T:input)
Begin
  If t>MaximaProfundidad Then
    Begin
      Write('Profundidad mayor a la permitida')
      Finalizar el programa
    End

    If  $1+M/L \geq p$  Then
      Return

  /*Accion A*/
  Procesar(L,M+t+3,t+1)

  /*Accion B*/
  Procesar(M+1,L+1,t+1)

End.
```

Al programa pueden agregarse funciones para que además de determinar si el árbol es finito, también lo imprima. Fue así como se generaron los árboles mostrados anteriormente para $p=1.7$ y $p=1.72$. Este programa fue ejecutado con diversos valores de p . Se determinó de esta manera que el árbol generado con $p=1.788$ es finito, siendo ésta la mejor cota inferior encontrada.

Lema 4.10: No existe algoritmo on-line c -competitivo con $c < 1.788$.

Demostración:

La demostración la provee el programa cuyo pseudo-código fue presentado antes, cuando el valor de p es 1.788.

En ese caso el programa termina indicando que el árbol es finito.

■

Cuando se intenta ejecutar el programa con $p=1.79$, el programa corre por mucho tiempo sin respuesta. En todo ese tiempo el programa no encontró una rama infinita, pues de haberla encontrado habría terminado indicando que encontró una rama más larga que la profundidad máxima permitida.

Nosotros creemos que las ramas más largas son aquellas que aplican la acción B siempre que ésta sea posible, e hicimos que el programa analice estas ramas primero. Por lo tanto, al no indicar el programa que encontró una rama de mayor profundidad

permitida significa que ha analizado (o está analizando) las ramas que consideramos mayores sin encontrar ninguna infinita.

Esto podría interpretarse como una ligera “evidencia” de que el árbol es finito. Por supuesto, existe la posibilidad de que exista una rama infinita aún no examinada. Por otro lado, como el árbol generado para $p \geq 1.79$ es enorme, es razonable pensar que analizarlo le llevará mucho tiempo, aún cuando el árbol sea finito.

En contrapartida, si $p \geq 1.85$ el programa finaliza rápidamente indicando una rama demasiado larga, aún cuando se le permiten longitudes muy grandes. Esto hace pensar que el valor mayor para p que produce un árbol finito está entre 1.79 y 1.85. Sin embargo, esto es sólo una conjetura.

¿Por qué creemos que las ramas mayores son aquellas que aplican B siempre que se puede? Esta conjetura parte del análisis realizado sobre los árboles finitos generados por el programa. Todos presentaban dicha característica, salvo por pequeñas perturbaciones causadas por las condiciones iniciales.

4.5.6 Cantidad de páginas necesarias

El principal inconveniente presentado por las instancias generadas por el adversario siguiendo el método descrito es que a medida que se desea obtener cotas inferiores mayores, la cantidad mínima de páginas distintas necesarias para generar las instancias se incrementa enormemente. Desde un punto de vista teórico este hecho no produce inconvenientes, pero desde un punto de vista práctico sería preferible obtener una cota inferior con instancias donde la cantidad de páginas diferentes sea menor.

La cantidad de páginas necesarias es por lo menos la cantidad de fases que se producen, pues en cada fase se hace por lo menos un pedido a una página nueva. La cantidad de fases es igual a la altura del árbol generado.

A partir del crecimiento de la altura del árbol, se puede ver cómo crece la cantidad de páginas distintas a medida que crece p . Se podría intentar disminuir la cantidad de páginas necesarias agregando al final de cada fase pedidos a páginas que no estén en R en vez de pedidos a páginas nuevas. De esta manera se podrían *reciclar* páginas previamente usadas. No hemos profundizado esta línea de razonamiento, pero de hacerse podría ayudar a reducir la cantidad de páginas necesarias.

5. Multi-Threaded Paging Semi-Online

5.1 Introducción

En este capítulo veremos una extensión del problema MTP, que llamaremos MTP Semi-Online. En esta extensión se suponen conocidas para el algoritmo on-line v secuencias de w secuencias, con $v \leq w$. Usaremos la notación $MTP(k, w, v)$ para referirnos a MTP Semi-Online. El problema $MTP(k, w)$ es el caso particular $MTP(k, w, 0)$.

Esta variante presenta especial interés cuando consideramos a MTP como un caso de secuenciamiento de tareas. En este escenario las secuencias de pedidos corresponden a secuencias de trabajos a realizar en un conjunto de máquinas, y cada page-fault indica una reconfiguración de la maquinaria. El objetivo es minimizar la cantidad de reconfiguraciones, las cuales se consideran mucho más costosas que la realización de las tareas. Como en este escenario pueden ser conocidas ciertas secuencias de tareas rutinarias, mientras que otras secuencias se determinan dinámicamente, el problema MTP Semi-Online es un modelo más fiel que el problema MTP.

En este capítulo analizaremos solamente el problema $MTP(1, 2, 1)$, donde la caché es de tamaño uno, se tienen dos secuencias de pedidos y una secuencia conocida. Mostraremos una cota inferior y una cota superior para este problema. La cota inferior encontrada es la razón áurea (ϕ), cuyo valor es aproximadamente 1.618. La cota superior encontrada es 2 y se hereda del problema $MTP(1, 2)$.

5.2 Definición del problema

El problema que analizaremos es una variante del problema $MTP(1, 2)$, donde se tienen dos secuencias y la caché es de tamaño uno. La diferencia es que en este caso el algoritmo on-line conoce una de las dos secuencias.

Por lo tanto, las instancias del problema están formadas por las secuencias S_1 y S_2 de pedidos a páginas. El algoritmo on-line debe decidir en qué orden servir los pedidos de ambas secuencias intentando minimizar la cantidad de page-faults incurridos en servir completamente las secuencias. Para decidir el siguiente pedido a servir puede basarse en la secuencia S_2 completa, los pedidos de la secuencia S_1 ya servidos, y el primer pedido aún no servido de la secuencia S_1 .

Como en este problema al algoritmo on-line dispone de mayor información que en el problema MTP, es de esperar obtener cotas inferiores y superiores para la competitividad de los algoritmos on-line más bajas. Hemos encontrado nuevas cotas inferiores, pero lamentablemente no hemos encontrado cotas superiores mejores que las provistas por el problema MTP.

Cualquier cota superior para el problema MTP(1,2) es válida también en MTP(1,2,1), pues todo algoritmo on-line para MTP(1,2) es también un algoritmo on-line para MTP(1,2,1). Por lo tanto obtenemos fácilmente la cota superior 2, proveniente de MTP.

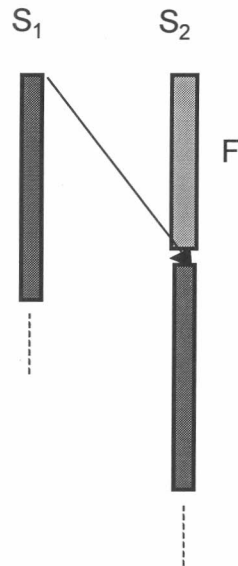
En MTP(1,2,1) el algoritmo on-line conoce una de las dos secuencias, disponiendo de más información que en el problema MTP(1,2). Consecuentemente es de esperar que pueda obtener mejores resultados. Por este motivo las cotas inferiores de MTP(1,2) no son necesariamente válidas en este problema. En la siguiente sección veremos la cota inferior encontrada.

5.3 Cota Inferior

Dado un algoritmo on-line A y un valor c tal que $1.5 \leq c < \phi$ el adversario puede generar una instancia I tal que el costo $C_A(I) / C_0(I)$ sea mayor o igual a c . Para lograr esto elige un número entero positivo n y construye la secuencia S_2 como la secuencia de pedidos a las páginas 1,2,3, ..., n . La secuencia S_1 la construye en fases.

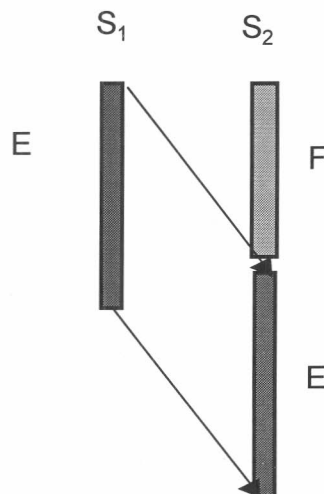
Sea F un entero positivo tal que $F > \frac{c-1}{1+c-c^2}$. En cada fase el adversario presenta en la secuencia S_1 los mismos pedidos que en la secuencia S_2 pero corridos F posiciones.

El siguiente diagrama ejemplifica esta construcción:



Sea E la cantidad de pedidos servidos por el algoritmo on-line de la secuencia S_1 hasta la primera vez que aparece en la fase.

En el siguiente diagrama se ve un caso en que el algoritmo on-line aparee dos pedidos habiendo servido $(E-1)$ pedidos de S_1 sin aparearlos. Las secuencias marcadas con la misma letra contienen los mismos pedidos.

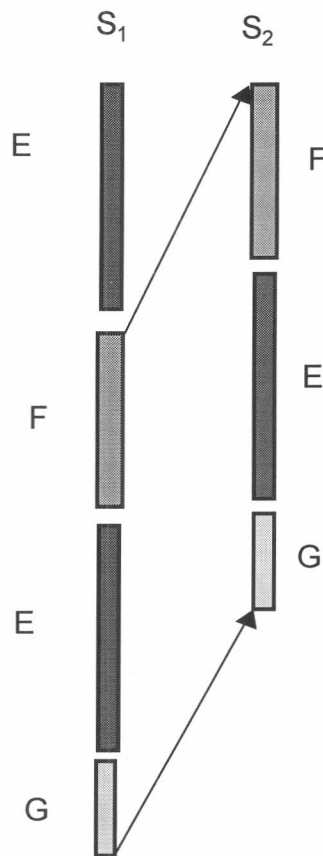


Caso 1) Si $1 + \frac{E-1}{F+E}$ el adversario termina la fase y comienza una fase nueva.

Caso 2) En caso contrario el adversario agrega una copia de F y E a la construcción de la secuencia S_1 . Como estos pedidos no se aparean con ningún pedido restante de la secuencia S_2 , el algoritmo on-line debe pagar por todos ellos.

Cada pedido servido de la secuencia S_2 antes de completar los pedidos de S_1 se agregan al final de la construcción de S_1 . La fase termina cuando el algoritmo on-line termina de

servir la construcción de S_1 . Llamemos G a la cantidad de pedidos que el algoritmo on-line sirve de S_2 antes de terminar la fase.



Sea el costo C_A del algoritmo on-line en la fase y C_0 el costo del óptimo off-line.

En el caso 1) C_A es $E-1 + F + E$ y C_0 es $E + F$.

Por lo tanto $C_A/C_0 = \frac{E-1 + F + E}{F+E} = 1 + \frac{E-1}{F+E}$.

Como en el caso 1) $1 + \frac{E-1}{F+E}$ debe ser mayor o igual a c , obtenemos $C_A/C_0 \geq c$.

En el caso 2) tenemos que:

C_A es $E-1 + F + E + F + E + G$ y C_0 es $E + F + E + G$.

Por lo tanto $C_A/C_0 = 1 + \frac{E-1 + F + G}{2E + F + G}$.

Veamos bajo qué condiciones C_A/C_0 es mayor o igual a c :

Es fácil ver que el valor de C_A/C_0 es menor cuando G es menor, por lo que el peor caso se da cuando G es igual a 0. Consecuentemente supongamos $G=0$ y $C_A/C_0 = 1 + \frac{E-1+F}{2E+F}$. Esto equivale a suponer que el algoritmo on-line sirve los pedidos de la construcción de S_1 consecutivamente.

$$\begin{aligned} C_A/C_0 \geq c &\Leftrightarrow 1 + \frac{E-1+F}{2E+F} \geq c \Leftrightarrow E+F-1 \geq (c-1)(2E+F) \Leftrightarrow \\ &\Leftrightarrow E+F-1 \geq 2cE+cF-2E-F \Leftrightarrow 3E-2cE \geq cF-2F+1 \Leftrightarrow \\ &\Leftrightarrow E(3-2c) \geq F(c-2)+1 \Leftrightarrow E(2c-3) \leq F(2-c)-1 \Leftrightarrow \\ &\Leftrightarrow E \leq \frac{F(2-c)-1}{(2c-3)}. \end{aligned}$$

Por otro lado, como estamos en el caso 2), $1 + \frac{E-1}{F+E}$ es menor que c , por lo que tenemos:

$$\begin{aligned} 1 + \frac{E-1}{F+E} < c &\Rightarrow E-1 < (c-1)(F+E) \Rightarrow E < cF+cE-F-E+1 \Rightarrow \\ &\Rightarrow 2E-cE < cF-F+1 \Rightarrow E(2-c) < F(c-1)+1 \Rightarrow \\ &\Rightarrow E < \frac{F(c-1)+1}{(2-c)}. \end{aligned}$$

Veamos que $\frac{F(c-1)+1}{(2-c)} \leq \frac{F(2-c)-1}{(2c-3)}$, por lo que $E < \frac{F(c-1)+1}{(2-c)} \leq \frac{F(2-c)-1}{(2c-3)} \Rightarrow$

$C_A/C_0 \geq c$:

$$\begin{aligned} \frac{F(c-1)+1}{(2-c)} &\leq \frac{F(2-c)-1}{(2c-3)} \Leftrightarrow \\ &\Leftrightarrow (F(c-1)+1)(2c-3) \leq (F(2-c)-1)(2-c) \Leftrightarrow \\ &\Leftrightarrow 2cF(c-1)+2c-3F(c-1)-3 \leq 2F(2-c)-2-cF(2-c)+c \Leftrightarrow \\ &\Leftrightarrow 2c^2F-2cF+2c-3cF+3F-13 \leq 4F-2cF-2-2cF+c^2F+c \Leftrightarrow \\ &\Leftrightarrow c^2F+c-cF-F-1 \leq 0 \Leftrightarrow \\ &\Leftrightarrow F(c^2-c-1)+c-1 \leq 0 \Leftrightarrow \\ &\Leftrightarrow F(c^2-c-1) \leq 1-c \Leftrightarrow \\ &\text{(tanto } c^2-c-1 \text{ como } 1-c \text{ son negativos pues } 1.5 \leq c < \phi) \\ &\Leftrightarrow F(1+c-c^2) \geq c-1 \Leftrightarrow \\ &\Leftrightarrow F \geq \frac{c-1}{(1+c-c^2)}. \text{ Esto es cierto por la restricción al elegir } F. \end{aligned}$$

Por lo tanto $\frac{F(c-1)+1}{(2-c)} \leq \frac{F(2-c)-1}{(2c-3)}$, por lo que $E \leq \frac{F(2-c)-1}{(2c-3)} \Rightarrow C_A/C_0 \geq c$.

Consecuentemente, en ambos casos C_A/C_0 es mayor o igual a c .

Lema 5.1: No existe un algoritmo on-line c -competitivo con $c < \phi$.

Demostración:

Elijamos c' tal que $c < c' < \phi$ y elijamos F tal que $F > \frac{c'-1}{1+c'-c'^2}$.

La construcción presentada anteriormente nos permite generar una instancia de costo arbitrariamente grande donde $C_A(I) \geq c' C_0(I) + b$, donde la constante b surge de los pedidos restantes en la última fase. El lema 2.4 nos garantiza que no puede existir ningún algoritmo on-line c -competitivo con $c < c' < \phi$.

■

6. Caché de tamaño uno y varias secuencias

6.1 Introducción

Como se vio en el capítulo 1, en los trabajos realizados por Feuerstein[1] y Feuerstein-Strejilevich de Loma[2] se demostró que la competitividad de ningún algoritmo on-line puede ser menor que $k+1-1/w$. En el caso particular tratado en este capítulo, asumiremos una caché de tamaño uno ($k=1$). Por lo tanto, la cota inferior se transforma en: $k+1-1/w = 2 - 1/w$. Se ve claramente que esta cota inferior es menor que 2 para todo w .

La cota superior encontrada en los mismos trabajos de Feuerstein[1] y Feuerstein-Strejilevich de Loma[2] es wk , que se convierte para nuestro caso particular en w . Hay una brecha entre ambas cotas: la cota superior crece linealmente con w mientras que la inferior está acotada superiormente por 2. A medida que w crece, la diferencia entre las cotas se agranda también.

En este capítulo intentaremos disminuir esta diferencia. Veremos por un lado cómo disponiendo de una cota superior para el problema MTP(1,2), podemos encontrar una cota superior para el caso $k=1$ y w arbitrario. Reducir la cota superior para MTP(1,2) implicaría reducirla también para MTP(1, w) con $w>2$. Esta generalización permitiría centrar el análisis en un caso muy particular ($k=1$, $w=2$) y obtener cotas para casos más generales ($k=1$, w arbitrario).

Por otro lado, hemos conseguido obtener cotas inferiores mayores que 2. Basándonos en una idea propuesta por Alborzi et al.[4] para un problema relacionado con el nuestro, pudimos idear un método para generar instancias que aseguren que el costo del algoritmo on-line sea por lo menos $\frac{\lfloor \log w \rfloor}{2} + 1$ veces el costo del algoritmo óptimo off-line.

La cota inferior encontrada crece logarítmicamente con w , mientras que la cota superior crece linealmente con w . La diferencia entre estas cotas es menor que la conocida anteriormente, donde la cota inferior estaba acotada por la constante 2.

6.2 Cotas superiores

En esta sección mostraremos cómo podemos utilizar algoritmos competitivos y cotas superiores obtenidas para el problema MTP(1,2), para obtener cotas para el problema MTP(1, w).

Llamaremos c_w a la cota superior encontrada para el caso de w secuencias. Veremos cómo esta cota se relaciona con las cotas $c_{w'}$ cuando w' es menor que w . Mostraremos que si se obtiene una buena cota superior c_2 , ésta puede ser usada para obtener buenas cotas c_w .

Presentaremos dos formas de obtener c_w a partir de c_2 . La primera que mostraremos fue encontrada en primer lugar, y luego fue mejorada para obtener la segunda. Ambas formas se basan en un buen valor para c_2 . Usando el valor conocido hasta ahora ($c_2=2$) las cotas encontradas obtienen un valor para c_w igual a w , que es el mismo valor encontrado en Feuerstein[1] y Feuerstein-Strejilevich de Loma[2]. Para un valor menor de c_2 las cotas encontradas serían estrictamente menores a w .

6.2.1 Un algoritmo on-line para varias secuencias

El algoritmo que presentaremos para servir w secuencias se basa en un algoritmo para servir $w-1$ secuencias.

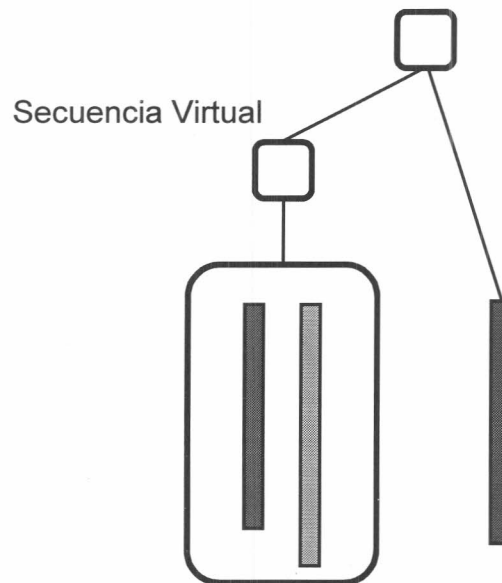
Sea A_{w-1} un algoritmo para $w-1$ secuencias y A_2 un algoritmo para dos secuencias.

Construyamos el algoritmo A_w para w secuencias utilizando los algoritmos A_{w-1} y A_2 :

- 1) A_{w-1} para procesar las primeras $w-1$ secuencias de la instancia. De estas $w-1$ secuencias surge una nueva secuencia virtual, dada por el orden en que el algoritmo A_{w-1} sirve los pedidos de las $w-1$ primeras secuencias.

- 2) A_2 para procesar la última secuencia de la instancia junto con la secuencia virtual formada por A_{w-1} .

Mostraremos el algoritmo sobre el siguiente esquema:



Inicialmente se utiliza A_{w-1} para decidir el primer pedido de la secuencia virtual. Ese pedido será a la página que el algoritmo A_{w-1} indica que debe servirse. Una vez que se conoce el primer pedido de la secuencia virtual, se utiliza A_2 para determinar qué pedido servir: el primero de la secuencia virtual o el primero de la última secuencia. Si el pedido a servir corresponde a la última secuencia, se sirve este pedido. Si corresponde a la secuencia virtual, se sirve este pedido y se utiliza el algoritmo A_{w-1} para determinar el siguiente pedido de la secuencia virtual.

Con el nuevo par de pedidos de la secuencia virtual y la última secuencia se vuelve a utilizar el algoritmo A_2 para determinar el siguiente pedido a servir.

Este proceso se reitera hasta servir todos los pedidos. Si en algún momento se termina de servir completamente la última secuencia mientras que aún quedan pedidos sin servir en la secuencia virtual, se sirven los pedidos restantes emulando al algoritmo A_{w-1} . Si la secuencia que se termina primero es la secuencia virtual, se sirven consecutivamente los pedidos restantes de la última secuencia.

Calculemos la competitividad obtenida por el algoritmo anteriormente descrito en función de c_{w-1} . Sea L_n la longitud de la secuencia n , $1 \leq n \leq w$ y Opt_{w-1} el costo de un algoritmo off-line óptimo que sirve las $w-1$ primeras secuencias. Llamemos C_o al costo del algoritmo óptimo off-line al servir las w secuencias y C_A al costo del algoritmo on-line descrito.

Lema 6.1: Dado un algoritmo A_{w-1} c_{w-1} -competitivo para el problema $MTP(1, w-1)$, y un algoritmo A_2 para el problema $MTP(1, 2)$, se puede construir un algoritmo A_w $(c_{w-1}+1)$ -competitivo para el problema $MTP(1, w)$.

Demostración:

El costo del algoritmo off-line al servir las w secuencias no puede ser menor a Opt_{w-1} ni a L_w , pues por lo menos debe servir la secuencia w y el costo de servir las primeras $w-1$ no puede ser mayor que el de servir las w secuencias. Por lo tanto:

$$C_0 \geq Opt_{w-1} \text{ y } C_0 \geq L_w$$

En el peor de los casos el algoritmo on-line no aparea ninguna página de la secuencia virtual con la última secuencia. El costo de servir las primeras $w-1$ secuencias es menor o igual a $c_{w-1} Opt_{w-1} + b$ (para alguna constante b independiente de la instancia), mientras que servir la última secuencia le cuesta L_w . Por lo tanto:

$$\begin{aligned} C_A &\leq c_{w-1} Opt_{w-1} + b + L_w && \Rightarrow \\ \Rightarrow C_A &\leq c_{w-1} C_0 + b + L_w && \Rightarrow \\ \Rightarrow C_A &\leq c_{w-1} C_0 + b + C_0 && \Rightarrow \\ \Rightarrow C_A &\leq (c_{w-1} + 1)C_0 + b \end{aligned}$$

En consecuencia, el algoritmo A_w es $(c_{w-1}+1)$ -competitivo.

■

Lema 6.2: Dado un algoritmo A_2 c_2 -competitivo para $MTP(1, 2)$, existe un algoritmo A_w (c_2+w-2) -competitivo para $MTP(1, w)$, todo $w \geq 2$.

Demostración:

Podemos aplicar el siguiente método para obtener el algoritmo A_w para w secuencias a partir del algoritmo A_2 :

A partir de A_2 obtenemos el algoritmo A_3 para tres secuencias usando el procedimiento descrito anteriormente. El algoritmo A_3 será entonces (c_2+1) -competitivo por el lema anterior.

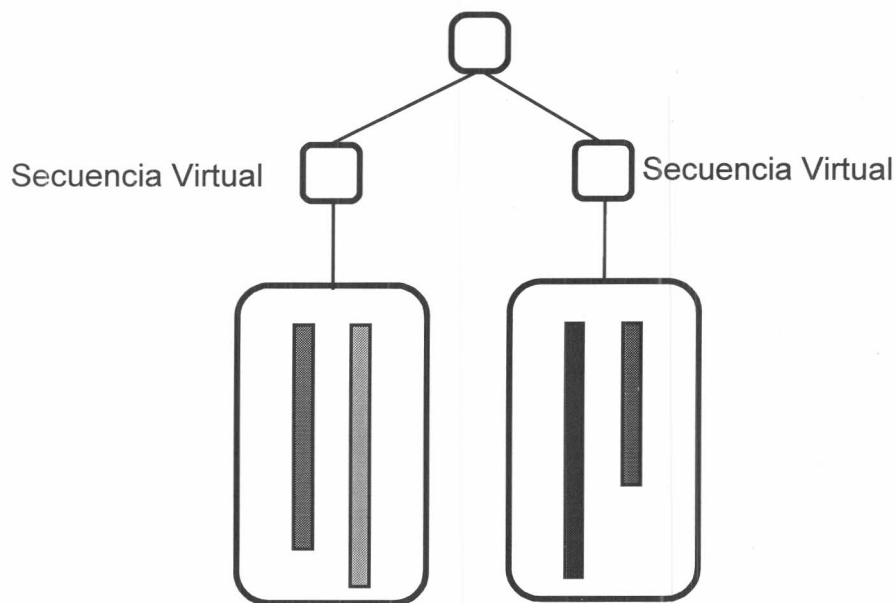
Luego a partir de A_3 obtenemos el algoritmo A_4 para cuatro secuencias. Este algoritmo será (c_2+2) -competitivo también por el lema anterior.

Inductivamente, podemos continuar esta construcción hasta obtener el algoritmo A_w para w secuencias. Este algoritmo será (c_2+w-2) -competitivo.

■

6.2.2 Un algoritmo *Divide And Conquer*

Podemos mejorar la cota para c_w encontrada anteriormente. Inicialmente nos restringiremos a valores de w que sean potencias de 2 y luego lo aplicaremos a valores w arbitrarios. La idea es dividir las w secuencias en dos grupos, cada uno con $w/2$ secuencias. A partir de cada grupo podemos obtener una secuencia virtual usando un algoritmo $A_{w/2}$ para $w/2$ secuencias. Finalmente servimos ambas secuencias virtuales usando un algoritmo A_2 para dos secuencias. El razonamiento es completamente análogo al de la sección anterior, salvo que en vez de separar las w secuencias en dos grupos, uno de $w-1$ secuencias y otro de una secuencia, se separan en dos grupos de $w/2$ secuencias cada uno.



Calculemos la competitividad obtenida por el algoritmo anteriormente descrito para w secuencias en función de $c_{w/2}$. Para el lema siguiente sólo necesitamos que w sea par. Sin embargo, posteriormente necesitaremos que w sea potencia de 2.

Lema 6.3: Dado un algoritmo $A_{w/2}$ $c_{w/2}$ -competitivo para el problema $MTP(1, w/2)$, se puede construir un algoritmo A_w $(2c_{w/2})$ -competitivo para el problema $MTP(1, w)$.

Demostración:

Llamemos Opt_1 al costo del algoritmo off-line óptimo para las $w/2$ primeras secuencias. Llamemos Opt_2 al costo del algoritmo off-line óptimo para las $w/2$ últimas secuencias. Sea L_n la longitud de la secuencia n , $1 \leq n \leq w$. Llamemos C_o al costo del algoritmo óptimo off-line y C_A al costo del algoritmo on-line construido según el procedimiento descrito.

El costo del algoritmo off-line no puede ser menor a Opt_1 ni a Opt_2 , pues el costo de servir $w/2$ secuencias no puede ser mayor que el de servir las w secuencias. Por lo tanto:

$$C_0 \geq Opt_1 \text{ y } C_0 \geq Opt_2$$

En el peor de los casos el algoritmo on-line no aparea ninguna página de las secuencias virtuales. El costo de servir las primeras $w/2$ secuencias es menor o igual a $c_{w/2} Opt_1 + b$ (para alguna constante b) y el costo de servir las últimas $w/2$ es menor o igual a $c_{w/2} Opt_2 + b$. Por lo tanto:

$$\begin{aligned} C_A &\leq c_{w/2} Opt_1 + b + c_{w/2} Opt_2 + b && \Rightarrow \\ \Rightarrow C_A &\leq c_{w/2} C_0 + b + c_{w/2} C_0 + b && \Rightarrow \\ \Rightarrow C_A &\leq 2c_{w/2} C_0 + 2b \end{aligned}$$

Por lo tanto el algoritmo A_w es $(2c_{w/2})$ -competitivo.

■

Lema 6.4: Si existe un algoritmo A_2 para 2 secuencias c_2 -competitivo, existe un algoritmo A_w para w secuencias $(2c_{w/2})$ -competitivo para todo $w=2^i$ todo entero $i \geq 2$.

Demostración:

Podemos aplicar el siguiente método para obtener el algoritmo A_w para w secuencias a partir del algoritmo A_2 :

A partir de A_2 obtenemos el algoritmo A_4 para cuatro secuencias usando el procedimiento descrito anteriormente. El algoritmo A_4 será entonces $(2c_2)$ -competitivo por el lema anterior. Luego a partir de A_4 obtenemos el algoritmo A_8 para ocho secuencias. Este algoritmo será $(4c_2)$ -competitivo también por el lema anterior.

Inductivamente, podemos continuar esta construcción hasta obtener el algoritmo A_w para w secuencias. Este algoritmo será $(\frac{w}{2}c_2)$ -competitivo.

■

Podemos ver que esta nueva cota es mejor o igual a la encontrada en la sección anterior. Suponemos $c_2 \leq 2$ pues como vimos en el capítulo 2 todo algoritmo lazy es 2-competitivo.

Lema 6.5: Con $w=2^i$ y $c_2 \leq 2$ se tiene: $\frac{w}{2}c_2 \leq c_2 + w - 2$, todo entero $i \geq 2$

Demostración:

$$\begin{aligned} \frac{w}{2} c_2 &\leq c_2 + w - 2 && \Leftrightarrow \\ \Leftrightarrow w c_2 &\leq 2c_2 + 2w - 4 && \Leftrightarrow \end{aligned}$$

$$\begin{aligned}
&\Leftrightarrow 4 - 2c_2 \leq w(2 - c_2) \quad \Leftrightarrow \\
&\Leftrightarrow (4 - 2c_2)/(2 - c_2) \leq w \quad \Leftrightarrow \\
&\Leftrightarrow 2 \leq w
\end{aligned}$$

■

Juntando ambas cotas superiores para c_w ($2c_{w/2}$ y $c_{w-1}+1$) podemos obtener una cota para el caso en el cual w no es una potencia de 2. La idea es que si w no es par, podemos obtener la cota $c_w \leq 2c_{(w-1)/2} + 1$, pues $c_{w-1} \leq 2c_{(w-1)/2}$ y $c_w \leq c_{w-1} + 1$. Si w es par, usamos la cota $c_w \leq 2c_{w/2}$.

En el caso donde w es impar la cota $c_w \leq 2c_{(w-1)/2} + 1$ corresponde a construir un algoritmo A_w a partir de un algoritmo A_{w-1} para $w-1$ secuencias, construido a su vez a partir de un algoritmo $A_{(w-1)/2}$ para $\frac{w-1}{2}$ secuencias.

En el caso donde w es par la cota $c_w \leq 2c_{w/2}$ corresponde a construir un algoritmo A_w a partir de un algoritmo $A_{w/2}$ para $w/2$ secuencias. Por lo tanto tenemos que resolver la recursión:

$$\begin{aligned}
f(w) &= 2f((w-1)/2) + 1 && \text{si } w \text{ es impar} \\
f(w) &= 2f(w/2) && \text{si } w \text{ es par}
\end{aligned}$$

partiendo de $f(2) = c_2$ y $f(3) = c_2 + 1$.

Usando esta recursión podemos afirmar que $c_w \leq f(w)$.

Lema 6.6: Sea $f(w)$ una función definida por la recursión:

$$\begin{aligned}
f(w) &= 2f((w-1)/2) + 1 && \text{si } w \text{ es impar} \\
f(w) &= 2f(w/2) && \text{si } w \text{ es par}
\end{aligned}$$

con $f(2) = c_2$ y $f(3) = c_2 + 1$.

Entonces $f(2^i + r) = 2^{i-1} c_2 + r$, con $0 \leq r < 2^i$.

Demostración:

Lo demostraremos por inducción.

Los casos base son:

$$\begin{aligned}
f(2) &= f(2^1 + 0) = 2^0 c_2 + 0 = c_2. \\
f(3) &= f(2^1 + 1) = 2^0 c_2 + 1 = c_2 + 1.
\end{aligned}$$

Suponiendo que para todo $w' < w$ la propiedad se cumple, veamos que se cumple para w :

Sea $w = 2^i + r$, $i \geq 2$ y $0 \leq r < 2^i$.

Si w es par (r debe ser par) tenemos:

$$f(w) = 2f(w/2) = 2f(2^{i-1} + r/2) = 2(2^{i-2} c_2 + r/2) = 2^{i-1} c_2 + r.$$

Si w es impar (r debe ser impar) tenemos:

$$\begin{aligned}
f(w) &= 2f((w-1)/2) + 1 = 2f(2^{i-1} + (r-1)/2) + 1 = \\
&= 2(2^{i-2} c_2 + (r-1)/2) + 1 = 2^{i-1} c_2 + (r-1) + 1 = 2^{i-1} c_2 + r.
\end{aligned}$$

■

Este lema nos permite dar la cota $c_{2^{i+r}} \leq 2^{i-1} c_2 + r$.

Corolario 6.7: Existe por lo menos un algoritmo on-line $(2^{i-1} c_2 + r)$ -competitivo para el problema MTP($1, 2^i + r$), para enteros $i \geq 2$ y $0 \leq r < 2^i$.

Demostración:

Se deduce directamente de los lemas anteriores.

■

6.3 Mejora de la cota inferior.

La cota inferior encontrada es $\frac{\lfloor \log w \rfloor}{2} + 1$. Esta cota inferior se obtiene a partir de una adaptación al problema MTP($1, w$) del resultado obtenido en [4] para el problema de k -clientes en el espacio métrico $\text{star}(w, 1)$. Este problema es similar al caso particular del MTP cuando $k=1$. Los resultados de esta sección son adaptaciones más o menos directas de sus resultados a este caso particular.

La cota inferior $\frac{\lfloor \log w \rfloor}{2} + 1$ contrasta con la cota inferior conocida anteriormente $2 - \frac{1}{w}$, pues esta última está acotada por la constante 2, independientemente del valor de w . Sin embargo, la nueva cota crece logarítmicamente en función de w .

En el caso particular $w=2$, se obtiene una cota inferior $\frac{\log w}{2} + 1 = \frac{\log 2}{2} + 1 = \frac{1}{2} + 1 = 1.5$. Esta cota inferior es menor a la mostrada en el capítulo 4, cuyo valor es 1.788.

6.3.1 Adaptación de resultados previos

En esta sección nos ocuparemos del caso $w=2^i$ para algún entero positivo i . Posteriormente ampliaremos los resultados a cualquier número de secuencias.

Mostraremos que para cada algoritmo on-line A existen instancias I de w secuencias donde A incurre en un costo $\frac{w \log w}{2} + w$ mientras que el algoritmo óptimo off-line incurre en un costo w . La relación entre ambos costos es $\left(\frac{w \log w}{2} + w \right) / w = \frac{\log w}{2} + 1$

También mostraremos que es posible obtener a partir de las instancias anteriores nuevas instancias tal que el costo off-line sea arbitrariamente grande y la relación entre los costos on-line y off-line sea también $\frac{\log w}{2} + 1$. Por lo tanto, el lema 2.4 nos garantiza que la competitividad de ningún algoritmo on-line puede ser menor que $\frac{\log w}{2} + 1$.

6.3.2 Generación de la instancia

La construcción de la instancia I se realiza en n fases, siendo $n = \log w + 1$. Inicialmente las w secuencias de I comienzan pidiendo una página distinta cada una. Llamemos w_f a la cantidad de secuencias de la instancia I que aún no se han finalizado de construir en la fase f . En cada fase f , estas w_f secuencias tendrán la misma longitud, que llamaremos L_f .

Cada fase finaliza cuando el algoritmo A sirve completamente todos los pedidos de $w_f/2$ secuencias. A medida que el algoritmo A va finalizando estas secuencias, el adversario las termina de construir. Por lo tanto, al finalizar una fase quedan $w_f/2$ secuencias aún sin terminar de construir, por lo que w_{f+1} es igual a $w_f/2$. Antes de comenzar la siguiente fase el adversario concatena a las $w_f/2$ secuencias no finalizadas una copia de los pedidos de las $w_f/2$ secuencias finalizadas, de una a una y en forma arbitraria. De esta manera, duplica la longitud de las $w_f/2$ secuencias restantes. Por lo tanto, $L_{f+1} = 2L_f$.

Sabiendo que $w_1 = w$ y que $L_1 = 1$, obtenemos que $w_f = w/2^{f-1}$ y que $L_f = 2^{f-1}$. Juntando ambos valores llegamos a que $w_f L_f = w$.

La construcción termina cuando $w_f = 1$, ya que no se pueden dividir las secuencias en dos grupos iguales. Esta condición se da cuando $w/2^{f-1} = 1 \Rightarrow w = 2^{f-1} \Rightarrow f = \log w + 1$. Por lo tanto hay $\log w + 1$ fases de construcción.

Veamos un ejemplo con $w = 8$. Al comienzo de la primera fase las secuencias contienen una página distinta cada una:

S_1	S_2	S_3	S_4	S_5	S_6	S_7	S_8
1	2	3	4	5	6	7	8

Las secuencias mostradas en la tabla anterior no se muestran completas. Indicaremos con un guión(-) al final de la secuencia cuando se muestre completa.

Supongamos que el algoritmo A sirve los pedidos a las páginas 1, 4, 3 y 2, en ese orden. Por lo tanto, el adversario finaliza la construcción de las primeras cuatro secuencias y las concatena al resto de las secuencias (Marcaremos las páginas servidas por el algoritmo en **negrita y cursiva**) :

S ₁	S ₂	S ₃	S ₄	S ₅	S ₆	S ₇	S ₈
1	2	3	4	5	6	7	8
-	-	-	-	1	2	3	4

En este momento comienza la segunda fase. Vemos como w_2 es 4 y L_2 es 2.

Supongamos que el algoritmo A sirve los pedidos a las páginas 5, 8, 6, 4 y 1 (estas páginas son arbitrarias y sólo a modo de ejemplo). En este punto el adversario ha finalizado las secuencias S_5 y S_8 , y puede concatenarlas a la parte construida de las secuencias S_6 y S_7 .

S ₁	S ₂	S ₃	S ₄	S ₅	S ₆	S ₇	S ₈
1	2	3	4	5	6	7	8
-	-	-	-	1	2	3	4
				-	5	8	-
					1	4	

Continuando con el ejemplo, en este punto comienza la fase 3. Vemos como w_3 es 2 y L_3 es 4. Supongamos que el algoritmo A sirve las páginas 2, 7, 3, 8, 5, 1. Por lo tanto se debe finalizar la secuencia S_6 y concatenar su contenido a la parte construida de la secuencia S_7 .

S ₁	S ₂	S ₃	S ₄	S ₅	S ₆	S ₇	S ₈
1	2	3	4	5	6	7	8
-	-	-	-	1	2	3	4
				-	5	8	-
					1	4	
					-	6	
						2	
						5	
						1	

Cuando el algoritmo A finalice la secuencia S_7 , el adversario termina esta secuencia dando por concluida la construcción de la instancia I.

S ₁	S ₂	S ₃	S ₄	S ₅	S ₆	S ₇	S ₈
1	2	3	4	5	6	7	8
-	-	-	-	1	2	3	4
				-	5	8	-
					1	4	
					-	6	
						2	
						5	
						1	
						-	

6.3.3 Análisis de la instancia

Como fue mostrado en [4], el algoritmo A falla al servir todos los pedidos. Esto se debe a que como al adversario concatena el contenido de las secuencias terminadas a las otras cuando terminan las fases, en ningún momento hay dos páginas iguales sin servir en la instancia I.

Consecuentemente su costo es igual a la cantidad total de pedidos en I. Para contarlos, observamos que la mitad de las secuencias tienen 1 pedido, 1/4 tienen 2 pedidos, 1/8 tienen 4 pedidos,..., 1 tiene $w/2$ pedidos y 1 tiene w pedidos. Por lo tanto, el costo de A

$$\text{es: } \sum_{j=1}^{\log w} 2^{j-1} \frac{w}{2^j} + w = \sum_{j=1}^{\log w} \frac{w}{2} + w = \frac{w \log w}{2} + w$$

Por construcción, el óptimo off-line puede servir la secuencia mayor (de w páginas) y servir todas las demás al mismo tiempo, incurriendo en un costo igual a w . Dividiendo la cota inferior para el costo de A por la cota superior del costo óptimo off-line, obtenemos que la relación entre los costos del algoritmo A y del algoritmo óptimo off-

$$\text{line no puede ser menor que } \frac{\frac{w \log w}{2} + w}{w} = \frac{\log w}{2} + 1.$$

6.3.4 Instancias de costo arbitrariamente grande

La construcción de I mostrada anteriormente tiene un costo óptimo off-line determinado. Esto se debe a que la longitud y los pedidos de las secuencias es fijo.

Se puede obtener un costo arbitrariamente grande si en la construcción anterior se reemplaza cada pedido por una secuencia de n pedidos a páginas todas distintas, donde n es un número positivo cualquiera. Cada pedido a la misma página es reemplazado por la misma secuencia de pedidos y no hay pedidos a páginas iguales en secuencias distintas. El adversario considera cada secuencia de n pedidos como una unidad.

Esto produce que la longitud de las secuencias se multipliquen por n , obteniendo un costo para el algoritmo on-line A igual a $n \frac{w \log w}{2} + nw$ mientras que el costo del algoritmo óptimo off-line no puede ser mayor a nw .

El cociente entre estos costos es mayor o igual a $\frac{\log w}{2} + 1$. Como el costo del algoritmo óptimo off-line es arbitrariamente grande y la relación entre los costos on-line y off-line está acotada inferiormente por $\frac{\log w}{2} + 1$, el lema 2.4 nos asegura que la competitividad de ningún algoritmo on-line puede ser menor que $\frac{\log w}{2} + 1$.

Esta construcción utiliza nw páginas distintas. Se puede obtener el mismo resultado con $2w$ páginas distintas si en vez de reemplazar cada página por una secuencia de n pedidos a páginas distintas, se reemplazan por una secuencia de pedidos a dos páginas que se alternan $n/2$ veces.

6.3.5 Adaptación a cualquier cantidad de secuencias

La cota inferior encontrada en los párrafos anteriores es válida si w es una potencia positiva de 2. Generalicemos este resultado para cualquier valor de w .

Lema 6.8: No existe algoritmo on-line para el problema $MTP(1, w)$ que sea c -competitivo con $c < \frac{\lfloor \log w \rfloor}{2} + 1$.

Demostración:

Sea $w = w' + r$, con $w' = 2^i$ y enteros $i \geq 1$ y $0 \leq r < 2^i$.

Sea I la instancia que tiene r secuencias vacías y w' secuencias construidas según la construcción de la sección anterior.

El costo del algoritmo on-line a sobre esta instancia es por lo menos $n \frac{w' \log w'}{2} + nw'$, y el costo del algoritmo óptimo off-line es a lo sumo nw' . El cociente entre los costos es por lo menos $\frac{\log w'}{2} + 1 = \frac{\lfloor \log w \rfloor}{2} + 1$.

Como esto es válido para todo valor positivo de n , el lema 2.4 nos asegura que la competitividad de ningún algoritmo on-line puede ser menor que $\frac{\lfloor \log w \rfloor}{2} + 1$.

■

7. Un problema off-line

7.1 Introducción

En este capítulo veremos un problema que está relacionado con MTP. Algunos resultados obtenidos serán utilizados en el capítulo siguiente para demostrar una cota inferior para el problema MPT en su caso más general: caché de cualquier tamaño y cualquier cantidad de secuencias.

El problema consiste en encontrar una cota superior al costo de servir v secuencias de longitud n de pedidos a páginas, cuando la caché es de tamaño k y las secuencias pueden tener pedidos de a lo sumo $k+1$ páginas distintas. Usamos la letra v en lugar de la tradicional w para indicar que no estamos trabajando en el problema on-line MTP, sino en un problema off-line relacionado.

El caso particular en que hay una sola secuencia es un problema clásico en paginación estándar. Este caso se refiere al costo máximo de servir una secuencia de n pedidos de a lo sumo $k+1$ páginas distintas teniendo una caché de tamaño k . Es fácil ver que se puede servir la secuencia en a lo sumo $\lceil n/k \rceil$ page-faults desalojando la página cuyo pedido es el más lejano.

La cota superior depende principalmente de la relación entre los valores v y n . El análisis realizado mostrará diferentes cotas para este problema en función de v , k y n . Como veremos en el próximo capítulo, este problema surge en un contexto donde el algoritmo on-line determina el valor de v mientras que el adversario determina el valor de n . El objetivo del adversario en ese contexto es servir las v secuencias con un costo proporcional a n e independiente de v . El resultado principal de este capítulo es mostrar que eligiendo n suficientemente grande, el adversario puede alcanzar su objetivo.

Por lo tanto, es de importancia la cota encontrada para este problema cuando n tiende a

infinito. Dicha cota es $\frac{e n}{k + \frac{e}{2}} = \frac{2 e n}{2k + e}$ donde e es la base del logaritmo natural. Esta cota

nos permitirá generalizar cualquier cota inferior para el problema MTP(1, w),

convirtiéndola en una cota para cualquier valor de k si se la multiplica por $\frac{k + \frac{e}{2}}{e} = \frac{k}{e} + \frac{1}{2}$.

7.2 Definición del problema

La definición del problema es la siguiente:

Dado el tamaño k de la caché y un número v de secuencias de n pedidos de a lo sumo $k+1$ páginas distintas, hallar cotas superiores al costo off-line de servir las v secuencias.

Si las secuencias contienen dos pedidos consecutivos a páginas iguales, estos pueden ser servidos conjuntamente sin costo adicional. Por lo tanto, supondremos secuencias duras.

Numeramos las $k+1$ páginas con los números $0, 1, \dots, k$.

Para especificar las condiciones iniciales, suponemos que la caché contiene al inicio las k páginas $1, 2, \dots, k$ y que todas las secuencias comienzan con un pedido a la página 0. Si comenzaran con un pedido a otra página, se podrían servir sin costo todos estos pedidos hasta lograr que el primer pedido no servido de cada secuencia sea un pedido a la página 0.

Si las condiciones iniciales fueran distintas, se podría alcanzar ese estado con un costo de a lo sumo k page-faults, necesarios para cargar la caché. Este costo es un valor acotado, independiente de n y de v , por lo que no lo consideramos más. A todas las cotas presentadas en este capítulo se debe sumar este valor si las condiciones iniciales no son las especificadas.

En la tabla siguiente vemos un ejemplo en el cual hay 4 secuencias ($v=4$) y una caché de tamaño tres ($k=3$):

4 secuencias de n filas

S_1	S_2	S_3	S_4
0	0	0	0
2	3	2	3
0	1	3	2
2	3	0	2

.....

3	3	2	1
1	2	3	0

La caché inicialmente contiene las páginas $1, \dots, k$:

1	2	3
---	---	---

7.3 Cotas para el caso infinito

El primer resultado que encontramos exige que las secuencias sean infinitas. En este caso no existe una cota superior al costo de servir todas las secuencias. En cambio, es interesante determinar la cantidad de páginas que se pueden servir en promedio por cada page-fault. Existe una relación intuitiva entre este valor y una cota superior al costo de servir las v secuencias en el caso de secuencias finitas. Cuantas más páginas sirvamos en promedio por cada page-fault, menor será el costo de servir las v secuencias completamente.

El siguiente lema nos muestra que se pueden servir por lo menos $\frac{v(k+1)}{2}$ páginas por cada page-fault, mientras que el lema posterior nos muestra que si v es par esta cota es óptima.

Lema 7.1: Si las secuencias son infinitas, se pueden servir por lo menos $\frac{v(k+1)}{2}$ pedidos por cada page-fault.

Demostración:

Mostraremos un método para servir por lo menos $\frac{v(k+1)}{2}$ pedidos por cada page-fault.

En todo momento en la caché hay k páginas. Sin pérdida de generalidad supongamos que las páginas cargadas son $1, 2, \dots, k$.

Si hay secuencias tal que el primer pedido no servido no es a la página 0, se sirven gratuitamente los pedidos hasta alcanzar en todas las secuencias un pedido a la página 0. Si alguna secuencia no presenta pedidos aún no servidos a esta página, se sirven indefinidamente los pedidos de esta secuencia, probando nuestro lema.

Por lo tanto podemos suponer que el primer pedido no servido de cada secuencia es a la página 0.

Definamos $d_{i,j}$ a la cantidad de pedidos no servidos anteriores al primer pedido no servido a la página j en la secuencia i , para $i=1\dots v$ y $j=1\dots k$.

Por ejemplo, supongamos que los primeros pedidos no servidos de las v secuencias son los mostrados en la tabla siguiente:

S_1	S_2	S_3	S_4
0	0	0	0
2	3	1	3
0	1	3	2
2	3	0	2
3	3	2	1
1	2	3	0

.....

En este caso, los valores para la matriz d son:

$$\begin{array}{lll}
 d_{1,1}=5 & d_{1,2}=1 & d_{1,3}=4 \\
 d_{2,1}=2 & d_{2,2}=5 & d_{2,3}=1 \\
 d_{3,1}=1 & d_{3,2}=4 & d_{3,3}=2 \\
 d_{4,1}=4 & d_{4,2}=2 & d_{4,3}=1
 \end{array}$$

Si algún valor de la matriz d esta indefinido es porque existe una secuencia que a partir de la posición actual no contiene ningún pedido a alguna página. Por lo tanto se pueden servir indefinidamente los pedidos de esta secuencia realizando un solo page-fault, probando nuestro lema.

Supongamos entonces que todos los valores de la matriz d están definidos. Dado que en cada secuencia i los pedidos a las páginas $1, \dots, k$ deben ocupar por lo menos los primeros k pedidos aún no servidos de la secuencia, tenemos que para $1 \leq i \leq v$:

$$\sum_{j=1}^k d_{ij} \geq 1+2+\dots+k = \sum_{j=1}^k j \Rightarrow \sum_{j=1}^k d_{ij} \geq \frac{k(k+1)}{2}$$

Como esta desigualdad es válida para todas las secuencias, podemos sumarlas:

$$\sum_{i=1}^v \sum_{j=1}^k d_{ij} \geq \sum_{i=1}^v \frac{k(k+1)}{2} \Rightarrow \sum_{i=1}^v \sum_{j=1}^k d_{ij} \geq \frac{vk(k+1)}{2}$$

Por otro lado, debe haber una página p tal que la suma de sus distancias en todas las secuencias sea por lo menos el valor promedio de la suma de todas las distancias:

$$\begin{aligned}
 \exists p / \sum_{i=1}^v d_{ip} &\geq \frac{\sum_{j=1}^k \sum_{i=1}^v d_{ij}}{k} \Rightarrow \sum_{i=1}^v d_{ip} \geq \frac{\frac{vk(k+1)}{2}}{k} \Rightarrow \sum_{i=1}^v d_{ip} \geq \frac{v(k+1)}{2} \\
 \Rightarrow \exists p / \sum_{i=1}^v d_{ip} &\geq \frac{v(k+1)}{2}
 \end{aligned}$$

Descartando la página p de la caché y cargando la página 0, se sirven d_{ip} pedidos de la secuencia S_i , dando un total de pedidos servidos igual a $\sum_{i=1}^v d_{ip}$. Por lo tanto descartando la página p se pueden servir por lo menos $\frac{v(k+1)}{2}$ pedidos.

Repitiendo indefinidamente este procedimiento se puede asegurar que en todo momento se pueden servir por lo menos $\frac{v(k+1)}{2}$ páginas por cada page-fault.

■

El siguiente lema nos muestra que si v es par, la cota encontrada en el lema anterior es óptima.

Lema 7.2: Si v es par existen instancias en las que no se pueden servir más de $\frac{v(k+1)}{2}$ pedidos por cada page-fault.

Demostración:

Sea $v=2m$ para algún entero positivo m .

Sea I la instancia formada por m pares de las secuencias infinitas:

$S_1: 0, 1, 2, \dots, k-1, k, 0, 1, 2, \dots$

$S_2: 0, k, k-1, k-2, \dots, 2, 1, 0, k, k-1, k-2, \dots$

Desalojando una página p de la caché y cargando la página 0 se pueden servir p pedidos de S_1 y $(k+1)-p$ pedidos de S_2 . Por lo tanto en los m pares de secuencias se sirven a lo sumo:

$$m(p + (k+1)-p) = m(k+1) = \frac{v(k+1)}{2}. \text{ Esto es válido para cualquier página } p.$$

Luego de servir los $\frac{v(k+1)}{2}$ pedidos, la caché contiene todas las páginas salvo p . Es fácil ver que la estructura de las secuencias S_1 y S_2 se mantiene luego de cada page-fault:

$S'_1: p, p+1, p+2, \dots, k-1, k, 0, 1, 2, \dots, p, p+1, \dots$

$S'_2: p, p-1, p-2, \dots, 2, 1, 0, k, k-1, k-2, \dots, p, p-1, \dots$

Por lo tanto, luego de desalojar cualquier página p , se pueden servir a lo sumo $\frac{v(k+1)}{2}$ pedidos y las secuencias presentan la misma estructura. Consecuentemente, nunca se pueden servir más de $\frac{v(k+1)}{2}$ pedidos por cada page-fault.

■

Corolario 7.3: Si v es impar existen instancias en las que no se pueden servir más de $\lceil \frac{v}{2} \rceil (k+1) + k$ pedidos por cada page-fault.

Demostración:

Sea $v=2m+1$ para algún entero positivo m .

Sea I la instancia formada por m pares de las secuencias infinitas S_1 y S_2 del lema anterior y una secuencia S_1 .

Luego de desalojar una página p , se pueden servir a lo sumo $m(k+1)$ pedidos de los m pares de secuencias, y a lo sumo k páginas de última secuencia S_1 . Por lo tanto no se pueden servir más de $m * (k+1) + k = \lceil \frac{v}{2} \rceil (k+1) + k$ pedidos.

Como la estructura de las secuencias se mantiene luego de cada desalojo, nunca se pueden servir más de $\lceil \frac{v}{2} \rceil (k+1) + k$ pedidos por cada page-fault.

■

El lema 7.2 nos muestra que para v par hemos encontrado la mejor cota posible. El siguiente lema nos da una cota útil para pequeños valores de v .

Lema 7.4: Se pueden servir por lo menos $v+k-1$ pedidos por cada page-fault.

Demostración:

Mostraremos un método para servir por lo menos $v+k-1$ pedidos por cada page-fault.

Se sirven los pedidos iterando por las secuencias. Al examinar la secuencia i , se determina la página p que aparece más lejana en esa secuencia. Se descarta la página p de la caché y se carga la primera página aún no servida de la secuencia i .

Luego se sirven todos los pedidos posibles de todas las secuencias sin costo alguno. Se sirven por lo menos k páginas en la secuencia i y una página en cada una de las $(v-1)$ secuencias restantes, dando un total de páginas servidas mayor o igual a $k+v-1$.

Una vez que se terminan de servir todos los pedidos que se pueden servir gratis, se pasa a la siguiente secuencia, en forma circular.

■

Como veremos más adelante, el algoritmo presentado en el lema anterior es útil para mostrar una cota en el caso de secuencias finitas. Su utilidad radica principalmente en que avanza relativamente parejo en todas las secuencias.

7.4 Cotas para el caso finito

Si con secuencias finitas fuera posible servir $\frac{v(k+1)}{2}$ pedidos por cada page-fault como en el caso infinito hasta finalizar todas las secuencias, podríamos obtener una cota superior del costo de servir todas las secuencias. La cantidad de pedidos total de las v secuencias de n pedidos es nv . Sirviendo $\frac{v(k+1)}{2}$ por cada page-fault, habría a lo sumo $\frac{vn}{\frac{v(k+1)}{2}} = \frac{2n}{k+1}$ page-faults. Por lo tanto una cota superior del costo sería $\frac{2n}{k+1}$.

Lamentablemente, cuando las secuencias se acaban no se puede garantizar que exista una página p tal que desalojándola de la caché se puedan servir $\frac{v(k+1)}{2}$ pedidos. El problema es que la forma de elegir la página p no garantiza que se avance en forma pareja en todas las secuencias. De esta manera algunas secuencias se terminan de servir antes que otras, desarticulando el método para elegir la página a desalojar.

Sin embargo, este resultado permite intuir que si n es grande, cuando las secuencias se empiecen a acabar ya se habrán realizado suficientes servicios de manera tal que servir el resto de las secuencias no influya demasiado en la tasa global de servicios por page-fault. Al final de esta sección veremos que para n suficientemente grande podemos garantizar un costo menor o igual a $\frac{en}{k+\frac{e}{2}}$. Es interesante notar la similitud entre este valor

$$\text{y } \frac{2n}{k+1}.$$

Comencemos con una cota válida para todo n y v .

Lema 7.5: Se pueden servir las v secuencias de n pedidos con a lo sumo n page-faults.

Demostración:

Probando que podemos servir por lo menos un pedido de cada secuencia por cada page-fault, estaríamos demostrando que con a lo sumo n page-faults se pueden servir las v secuencias. Veamos que esto es posible.

Llamemos *fila i* al conjunto de pedidos formado por los v pedidos en la posición i de cada secuencia. Dado que cada secuencia contiene pedidos de un conjunto de $k+1$ páginas, cada fila contiene a lo sumo $k+1$ páginas distintas.

El algoritmo que daremos en cada paso sirve una fila incurriendo en a lo sumo un page-fault.

En cada paso en la caché se encuentran todas las páginas salvo una, que llamaremos p . En la fila actual puede aparecer o no un pedido a la página p .

Si no aparece, el algoritmo sirve la fila entera sin ningún page-fault.

Si aparece, el algoritmo sirve todos los pedidos de la fila distintos de p . Luego desaloja de la caché una página elegida arbitrariamente y carga la página p . Finalmente sirve todos los pedidos restantes de la fila a la página p .

De esta manera el algoritmo sirvió toda la fila con a lo sumo un page-fault. Repitiendo este proceso por cada fila podemos servir todas las secuencias con a lo sumo n page-faults.

■

El lema anterior es una primera aproximación. Nos brinda una forma de servir todas las secuencias en a lo sumo n page-faults. El siguiente lema nos muestra que en algunos casos esta cota es óptima.

Lema 7.6: Si $v \geq k^{n-1}$, existen instancias para las que son necesarios n page-faults para servir todas las secuencias.

Demostración:

Recordemos que las v secuencias comienzan con 0 y la caché contiene inicialmente las k páginas $1 \dots k$.

Asumimos que $v = k^{n-1}$. Si $v > k^{n-1}$ suponemos que las últimas $v - k^{n-1}$ secuencias son idénticas a la primera.

Supongamos que los pedidos de las v secuencias forman todas las combinaciones posibles de longitud n de $k+1$ páginas, comenzando con 0 y sin tener 2 pedidos consecutivos a páginas iguales.

Hay k^{n-1} combinaciones distintas, pues la página i -ésima ($i > 1$) de la secuencia puede ser cualquier página salvo la anterior, por lo que hay k posibilidades para elegir. Como la primera página es 0, hay k^{n-1} combinaciones distintas de longitud n .

Como la cantidad de combinaciones es exactamente k^{n-1} , esta construcción define completamente las v secuencias.

Sea s la cantidad de page-faults mínima necesaria para servir las v secuencias completamente. En cada page-fault se desaloja una página de la caché y se carga otra. Todas las páginas desalojadas forman una secuencia de longitud s que llamaremos S .

Podemos ver la secuencia S como una indicación de la página a desalojar en cada momento.

Sabemos que s no puede ser mayor a n por el lema anterior. Supongamos $s < n$. Sea r una secuencia de pedidos formada por un pedido a la página 0 seguida por la secuencia S , y finalmente con una secuencia arbitraria de $n-s-1$ pedidos ($n-s-1$ podría ser 0). La longitud de r es $1+s+(n-s-1) = n$. Dado que las v secuencias son todas las posibles de longitud n , r debe ser alguna de ellas.

Sirviendo los pedidos de acuerdo a S , se sirven a lo sumo s páginas de la secuencia r . Esto se debe a que en cada page-fault se desaloja precisamente la página siguiente de r , por lo que se puede servir a lo sumo un pedido de r por cada page-fault. Por lo tanto, luego de s page-faults restan servir $n-s$ pedidos de la secuencia r . Como supusimos $s < n$, $n-s$ debe ser mayor a 0, por lo que la secuencia r aún no se terminó de servir. Esto es absurdo pues servir los pedidos de acuerdo a S permite servir todos los pedidos de las v secuencias.

El absurdo anterior provino de suponer $s < n$. Como $s \leq n$ por el lema 7.5, $s = n$.

■

El siguiente lema es una adaptación del lema 7.4 para el caso de n finito.

Lema 7.7: Se pueden servir las v secuencias de n pedidos con a lo sumo $v \lceil \frac{n}{k+v-1} \rceil$ page-faults.

Demostración:

Usemos el mismo algoritmo que en la demostración del lema 7.4.

En un ciclo completo del algoritmo se sirven por lo menos $k+v-1$ pedidos de cada secuencia: k pedidos cuando la secuencia es seleccionada y un pedido cada una de las $v-1$ veces que no es seleccionada.

La cantidad de page-faults en un ciclo completo es v .

Como en cada ciclo se sirven por lo menos $k+v-1$ pedidos de cada secuencia, con $\lceil \frac{n}{k+v-1} \rceil$ ciclos deben servirse todas las secuencias, por lo que la cantidad total de page-faults no puede exceder $v \lceil \frac{n}{k+v-1} \rceil$.

■

Analicemos esta cota para los valores $v=1$ y $v=2$:

- 1) Cantidad de secuencias igual a 1: Si $v=1$ la cota se transforma en $\lceil n/k \rceil$. Esta cota es óptima según se mostró en [2] y una instancia que obtiene esta cota está formada por la secuencia:

$S_1 = 0, 1, 2, \dots, k, 0, 1, \dots, k, 0, 1 \dots$ hasta completar n pedidos.

2) Cantidad de secuencias igual a 2: Si $v=2$ la cota se transforma en $2^{\lceil \frac{n}{k+1} \rceil}$. Esta cota también es óptima como muestra la instancia formada por el siguiente par de secuencias:

$S_1 = 0, 1, 2, \dots, k, 0, 1, \dots, k, 0, 1 \dots$ hasta completar n pedidos.

$S_2 = 0, k, k-1, \dots, 2, 1, 0, k, k-1, \dots, 2, 1, 0, k, \dots$ hasta completar n pedidos.

Veamos ahora la cota superior que usaremos en el siguiente capítulo para mejorar la cota inferior conocida del problema $MTP(k, w)$. Mostraremos que existe un valor de n tal que se puedan servir v secuencias de n pedidos con a lo sumo $\lceil \frac{e n}{k+e/2} \rceil$ page-faults, siendo e la base del logaritmo natural. Para obtener este resultado necesitamos definir el concepto de estrategia y mostrar algunos resultados previos.

7.4.1 Estrategias

Una estrategia es una secuencia de páginas que no comienza con la página 0 y no contiene 2 páginas consecutivas iguales. Podemos relacionar una estrategia con una secuencia de páginas que un algoritmo desaloja de la caché al servir los pedidos de una instancia del problema.

Dada una estrategia y una instancia del problema definimos un algoritmo A de la siguiente manera:

El algoritmo A funciona iterativamente. En el i -ésimo ciclo sirve todos los pedidos a páginas que se encuentran en su caché, en un orden arbitrario. Luego determina la página p que no se encuentra en su caché y la carga desalojando la i -ésima página de la estrategia.

El algoritmo termina cuando se sirvieron todos los pedidos de la instancia o se realizaron tantos ciclos como la longitud de la estrategia. La cantidad de page-faults realizados por el algoritmo A es menor o igual a la longitud de la estrategia. Entonces, si se sirvieron todos los pedidos de la instancia, el costo del algoritmo en servirlos fue menor o igual a la longitud de la estrategia.

Decimos que una estrategia e sirve una instancia I si el algoritmo definido anteriormente para la estrategia e y la instancia I sirve todos los pedidos de la instancia I . En caso contrario decimos que la estrategia e no sirve la instancia I .

Lema 7.8: Existen k^l estrategias distintas de longitud l .

Demostración:

La primera página de una estrategia e no puede ser 0 por definición. Por lo tanto hay k posibilidades para la primera página de e . La página i -ésima ($i > 1$) de la estrategia puede ser cualquier página salvo la anterior, por lo que también hay k posibilidades para elegir. Por lo tanto, hay k^l estrategias distintas de longitud l .
■

Veamos cuántas de estas estrategias no sirven una instancia dada.

Lema 7.9: Hay a lo sumo $v \binom{n}{l}$ estrategias de longitud l que no sirven una instancia I de v secuencias de longitud n de $k+1$ páginas distintas.

Demostración:

Llamemos S_i a la i -ésima secuencia de la instancia I .

Para cada estrategia e llamemos A_e al algoritmo relacionado con la estrategia e y la instancia I .

Cada vez que A_e desaloja una página de la caché, sirve los pedidos de cada secuencia hasta encontrar un pedido a la página desalojada. Por lo tanto, si A_e no sirve a la instancia I , es porque se encontró l veces con un pedido a la página descartada en alguna secuencia de I , una vez por cada página de la estrategia e . Esto significa que si A_e no sirve a la instancia I , es porque alguna secuencia de I contiene como subsecuencia de longitud l a la estrategia e .

La máxima cantidad de subsecuencias posibles de longitud l en una sola secuencia de longitud n es $\binom{n}{l}$. Suponiendo el peor caso, los conjuntos de subsecuencias de longitud n de cada secuencia de la instancia I son disjuntos. Consecuentemente, no puede haber más de $v \binom{n}{l}$ subsecuencias distintas en toda la instancia I . Si A_e no sirve la instancia I , la estrategia e debe ser alguna de estas subsecuencias. Por lo tanto, hay a lo sumo $v \binom{n}{l}$ estrategias de longitud l que no sirven una instancia I .
■

7.4.2 Cota superior

En la sección anterior mostramos que hay k^l estrategias distintas de longitud l . También mostramos que hay a lo sumo $v \binom{n}{l}$ estrategias de longitud l que no sirven completamente una instancia I . Por lo tanto, mostrando que $v \binom{n}{l} < k^l$, estaríamos mostrando que existe por lo menos una estrategia de longitud l que sirve las v secuencias de I , por lo que existe un algoritmo que sirve la instancia I con un costo menor o igual a l . Como queremos encontrar una cota superior al costo que sea lineal con respecto a n , hagamos $l = \alpha n$, con α independiente de n . El objetivo principal es lograr un valor de α que dependa solamente de k . Como veremos en los siguientes lemas, podemos lograr que $v \binom{n}{l} \leq k^l$ si hacemos n grande y elegimos un valor apropiado para α .

Mostraremos 2 lemas. El segundo provee un mejor resultado que el primero. Sin embargo, el primero es mucho más simple y muestra la forma en que fuimos mejorando las cotas encontradas. Notar que los logaritmos usados son todos en base 2.

Lema 7.10: Existe un valor de n tal que se pueda servir una instancia I de v secuencias de n pedidos con a lo sumo $\lceil n/\log k \rceil$ page-faults.

Demostración:

Alcanza con probar que existe un n tal que $v \binom{n}{l} < k^l$. Esto es igual a probar que $\frac{\binom{n}{l}}{k^l} < \frac{1}{v}$.

Por lo tanto, probando que $\lim_{n \rightarrow \infty} \frac{\binom{n}{l}}{k^l} = 0$ estaríamos probando el lema.

Como $l = \alpha n \Rightarrow$

$$\begin{aligned} \frac{\binom{n}{l}}{k^l} &= \frac{\binom{n}{\alpha n}}{k^{\alpha n}} = \frac{\binom{n}{\alpha n}}{(k^{\alpha/2} + k^{\alpha/2})^n} = \frac{\binom{n}{\alpha n}}{\sum_{i=0}^n \binom{n}{i} (k^{\alpha/2})^i (k^{\alpha/2})^{n-i}} = \\ &= \frac{\binom{n}{\alpha n}}{\sum_{i=0}^n \binom{n}{i} (k^{\alpha/2})^n} \leq \frac{\binom{n}{\alpha n}}{\binom{n}{\alpha n} (k^{\alpha/2})^n} = \frac{1}{(k^{\alpha/2})^n} \end{aligned}$$

Si $k^{\alpha/2} > 1$, se ve claramente que $\lim_{n \rightarrow \infty} \frac{1}{(k^{\alpha/2})^n} = 0$. En este caso, como $\frac{\binom{n}{l}}{k^l} > 0 \forall n$, la sucesión $\frac{\binom{n}{l}}{k^l}$ está acotada por dos sucesiones cuyo límite cuando $n \rightarrow \infty$ es 0. Por lo tanto el $\lim_{n \rightarrow \infty} \frac{\binom{n}{l}}{k^l}$ existe y es 0.

Como $k^{\alpha/2} > 1 \Leftrightarrow k^{\alpha} > 2 \Leftrightarrow \alpha \log k > \log 2 \Leftrightarrow \alpha \log k > 1 \Leftrightarrow \alpha > 1/\log k \Leftrightarrow l > n/\log k$ tenemos que si l es mayor a $n/\log k$, entonces $\lim_{n \rightarrow \infty} \frac{\binom{n}{l}}{k^l} = 0$. Por lo tanto existe una estrategia de longitud $\lceil n/\log k \rceil$ que sirve las v secuencias. Esto implica que existe un algoritmo que sirve las v secuencias con un costo menor o igual a $\lceil n/\log k \rceil$. ■

Finalmente podemos mostrar el lema que nos provee la cota que usaremos en el capítulo siguiente. Este lema mejora la cota encontrada en el lema anterior.

Lema 7.11: Existe un valor de n tal que se pueda servir una instancia I de v secuencias de n pedidos con a lo sumo $\lceil \frac{e \cdot n}{k+e/2} \rceil$ page-faults.

Demostración:

Si $k=1$ se sirve la instancia I con $n < \lceil \frac{e \cdot n}{1+e/2} \rceil$ page-faults, por lo que la cota es cierta.

Cuando $k > 1$ es suficiente probar que si $l = \alpha n$, con $\alpha = \frac{\lceil \frac{e \cdot n}{k+e/2} \rceil}{n}$ entonces $\lim_{n \rightarrow \infty} \frac{\binom{n}{l}}{k^l} = 0$. Notar que con $n > 1$, se verifica que $\frac{e}{k+e/2} < \alpha < 1$.

Usemos la aproximación de Stirling que nos dice que:

$x! = x^x e^{-x} \sqrt{x} G(x)$, con la propiedad de que $\lim_{x \rightarrow \infty} G(x) = g$, siendo g una constante. Usando esta aproximación, tenemos que:

$$\frac{\binom{n}{l}}{k^l} = \frac{n^n e^{-n} \sqrt{n} G(n)}{l^l e^{-l} \sqrt{l} G(l) \cdot (n-l)^{n-l} e^{-(n-l)} \sqrt{n-l} G(n-l)} =$$

$$\begin{aligned}
&= \frac{n^n \sqrt{n}}{l! \sqrt{l} \cdot (n-l)^{n-l} \sqrt{n-l}} H(n) \text{ [definiendo } H(n) = \frac{G(n)}{G(\alpha n) G(n-\alpha n)}] = \\
&= \frac{n^n \sqrt{n}}{\alpha n^{\alpha n} \sqrt{\alpha n} \cdot (n-\alpha n)^{n-\alpha n} \sqrt{n-\alpha n}} H(n) = \\
&= \frac{n^n \sqrt{n}}{\alpha n^{\alpha n} \sqrt{\alpha n} \cdot ((1-\alpha)n)^{(1-\alpha)n} \sqrt{(1-\alpha)n}} H(n) = \\
&= \frac{n^n}{\alpha^{\alpha n} n^{\alpha n} \sqrt{\alpha} \cdot (1-\alpha)^{(1-\alpha)n} n^{(1-\alpha)n} \sqrt{1-\alpha} \sqrt{n}} H(n) = \\
&= \frac{1}{\sqrt{\alpha(1-\alpha)n}} \left(\frac{1}{\alpha^{\alpha n} \cdot (1-\alpha)^{(1-\alpha)n}} \right) H(n) = \\
&= \frac{1}{\sqrt{\alpha(1-\alpha)n}} \left(\frac{1}{[\alpha^\alpha (1-\alpha)^{(1-\alpha)}]^n} \right) H(n).
\end{aligned}$$

$$\begin{aligned}
\text{Por lo tanto, } \lim_{n \rightarrow \infty} \frac{\binom{n}{l}}{k^l} &= \lim_{n \rightarrow \infty} \frac{1}{\sqrt{\alpha(1-\alpha)n}} \left(\frac{1}{[\alpha^\alpha (1-\alpha)^{(1-\alpha)} k^\alpha]^n} \right) H(n) = \\
&= \lim_{n \rightarrow \infty} \frac{1}{\sqrt{\alpha(1-\alpha)}} \lim_{n \rightarrow \infty} \frac{1}{\sqrt{n} [\alpha^\alpha (1-\alpha)^{(1-\alpha)} k^\alpha]^n} \lim_{n \rightarrow \infty} H(n) = \\
&= h \lim_{n \rightarrow \infty} \frac{1}{\sqrt{n} [\alpha^\alpha (1-\alpha)^{(1-\alpha)} k^\alpha]^n} \text{ siendo } h \text{ una constante.}
\end{aligned}$$

Si $\alpha^\alpha (1-\alpha)^{(1-\alpha)} k^\alpha \geq 1 \quad \forall n > 1$, se obtiene que:

$$\begin{aligned}
0 &\leq \lim_{n \rightarrow \infty} \frac{1}{\sqrt{n} [\alpha^\alpha (1-\alpha)^{(1-\alpha)} k^\alpha]^n} \leq \lim_{n \rightarrow \infty} \frac{1}{\sqrt{n}} = 0 \Rightarrow \\
\lim_{n \rightarrow \infty} \frac{1}{\sqrt{n} [\alpha^\alpha (1-\alpha)^{(1-\alpha)} k^\alpha]^n} &= 0 \Rightarrow \\
\lim_{n \rightarrow \infty} \frac{\binom{n}{l}}{k^l} &= 0
\end{aligned}$$

Para mostrar que $\alpha^\alpha (1-\alpha)^{(1-\alpha)} k^\alpha \geq 1 \quad \forall n > 1$, recordemos que $\frac{e}{k+e/2} < \alpha < 1 \Rightarrow$

$$\begin{aligned}
k &\geq e/\alpha - e/2 = e(1/\alpha - 1/2) = e \frac{1-\alpha/2}{\alpha} \\
\Rightarrow \alpha^\alpha (1-\alpha)^{(1-\alpha)} k^\alpha &\geq \alpha^\alpha (1-\alpha)^{(1-\alpha)} \left(e \frac{1-\alpha/2}{\alpha} \right)^\alpha \\
\Rightarrow \alpha^\alpha (1-\alpha)^{(1-\alpha)} k^\alpha &\geq \alpha^\alpha (1-\alpha)^{(1-\alpha)} e^\alpha (1-\alpha/2)^\alpha \alpha^{-\alpha}
\end{aligned}$$

$$\Rightarrow \alpha^\alpha (1-\alpha)^{(1-\alpha)} k^\alpha \geq (1-\alpha)^{(1-\alpha)} e^\alpha (1-\alpha/2)^\alpha$$

Es fácil ver que $(1-\alpha)^{(1-\alpha)} e^\alpha (1-\alpha/2)^\alpha \geq 1 \Leftrightarrow e \geq (1-\alpha)^{1-1/\alpha} + e\alpha/2$:

$$\begin{aligned} (1-\alpha)^{(1-\alpha)} e^\alpha (1-\alpha/2)^\alpha &\geq 1 && \Leftrightarrow \\ \Leftrightarrow (1-\alpha)^{1/\alpha-1} e (1-\alpha/2) &\geq 1 && \Leftrightarrow \\ \Leftrightarrow (1-\alpha)^{1/\alpha-1} (e - e\alpha/2) &\geq 1 && \Leftrightarrow \\ \Leftrightarrow (e - e\alpha/2) &\geq (1-\alpha)^{1-1/\alpha} && \Leftrightarrow \\ \Leftrightarrow e &\geq (1-\alpha)^{1-1/\alpha} + e\alpha/2 \end{aligned}$$

En el apéndice A se encuentra una demostración de que $(1-\alpha)^{1-1/\alpha} + e\alpha/2 < e$ para $0 < \alpha < 1$. Por lo tanto, $(1-\alpha)^{(1-\alpha)} e^\alpha (1-\alpha/2)^\alpha \geq 1 \Rightarrow \alpha^\alpha (1-\alpha)^{(1-\alpha)} k^\alpha \geq 1 \Rightarrow \lim_{n \rightarrow \infty} \frac{\binom{n}{l}}{k^l} = 0$.

A partir de este límite obtenemos que existe un valor de n y un valor de α tan cercano a $\frac{e}{k+e/2}$ como se desee tal que $v\left(\frac{n}{l}\right) < k^l$, finalizando la demostración. ■

Este último lema nos dice que si n es suficientemente grande, debe existir una estrategia de longitud $\lceil \frac{e \cdot n}{k+e/2} \rceil$ que sirva la instancia I. Por lo tanto existe un algoritmo que sirve la instancia I con un costo menor o igual a $\lceil \frac{e \cdot n}{k+e/2} \rceil$. Este resultado será usado en el capítulo siguiente para mostrar una cota inferior para el problema $MTP(k, w)$.

8. Caché de tamaño arbitrario

8.1 Introducción

En este capítulo veremos cómo podemos generar algoritmos wk -competitivos para el problema $MTP(k, w)$ a partir de cualquier algoritmo k -competitivo para paginación estándar.

También veremos cómo se puede utilizar una cota inferior obtenida para el problema MTP donde el tamaño de la caché es uno, y obtener una cota para el caso general.

Luego usaremos la cota encontrada en el capítulo 6 para obtener una nueva cota para el problema MTP en su forma más general.

8.2 Algoritmos kw -competitivos para MTP

En esta sección veremos una adaptación del lema 3.1, el cual supone una caché de tamaño uno.

Mostraremos que todo algoritmo para el problema de paginación estándar k -competitivo puede ser extendido en un algoritmo wk -competitivo para el problema $MTP(k, w)$. Notar que los algoritmos on-line k -competitivos son óptimos en paginación estándar. Un ejemplo de estos algoritmos es “Least recently used” (LRU).

Lema 8.1: A partir de un algoritmo on-line A para paginación estándar k -competitivo se puede generar un algoritmo A' para el problema $MTP(k, w)$ wk -competitivo.

Demostración:

Sea A' el algoritmo que sirve una a una las w secuencias, sirviendo cada secuencia usando el algoritmo A . Sirve completamente una secuencia antes de servir algún pedido de otra.

Dada una instancia I , llamemos $C_A(S_i)$ al costo del algoritmo A' al servir la secuencia S_i de la instancia I y $C_o(S_i)$ el costo del algoritmo óptimo off-line el servir la secuencia S_i .

Como el algoritmo A' sirve cada secuencia emulando el algoritmo A , y este algoritmo es k -competitivo, tenemos que para $1 \leq i \leq w$:

$$C_A(S_i) \leq k C_o(S_i) + b, \text{ para alguna constante } b \text{ dependiente del algoritmo } A.$$

Por lo tanto el costo del algoritmo A' en servir todas las secuencias es:

$$C_{A'}(I) = \sum_{i=1}^w C_A(S_i) \leq \sum_{i=1}^w (k C_o(S_i) + b) = w.b + k \sum_{i=1}^w C_o(S_i)$$

El costo del algoritmo óptimo off-line al servir todas las secuencias no puede ser menor a $\max(C_o(S_i))$:

$$C_o(I) \geq \max(C_o(S_i)).$$

Por lo tanto:

$$C_{A'}(I) \leq w.b + k \sum_{i=1}^w C_o(S_i) \leq w.b + k \sum_{i=1}^w \max(C_o(S_i)) \Rightarrow$$

$$C_{A'}(I) \leq w.b + k \sum_{i=1}^w C_o(I) = w.b + w.k.C_o(I) \Rightarrow$$

$$C_{A'}(I) \leq w.k.C_o(I) + b' \text{ para alguna constante } b' \text{ independiente de la instancia } I.$$

Como esta desigualdad es válida para toda instancia I , el algoritmo A' es wk -competitivo.

■

8.3 Generalización de cotas inferiores con caché de tamaño uno

En esta sección veremos que si d es una cota inferior para el problema $MTP(1, w)$, en el cual el tamaño de la caché es uno, entonces $\left(d \frac{k+e/2}{e+\varepsilon}\right)$ lo es para $MTP(k, w)$, con $k > 1$.

En la próxima sección utilizaremos este resultado junto con la cota inferior $\frac{\lfloor \log w \rfloor}{2} + 1$ para $MTP(1, w)$ encontrada en el capítulo 6, y mostraremos una cota inferior para el problema $MTP(k, w)$.

Este resultado buscado se obtiene generando para cada algoritmo A una instancia I tal que $C_A(I) > \left(d \frac{k+e/2}{e+\varepsilon}\right) C_o(I) + b$, para toda constante b y ε tan chico como se desee.

Lema 8.2: Si d es una cota inferior para el problema $MTP(1, w)$, entonces no existe un algoritmo on-line A para el problema $MTP(k, w)$ c -competitivo con $c < d \frac{k+e/2}{e}$.

Demostración:

Llamemos U el conjunto de páginas consideradas por el adversario. Supongamos que el adversario particiona este conjunto en clases de $k+1$ páginas distintas. Llamemos U' el conjunto cociente de U dada esta partición. Por lo tanto, cada elemento de U' es un conjunto de $k+1$ páginas, que llamaremos *super-página*.

Cada secuencia de la instancia I generada por el adversario está formada por bloques de l pedidos. Cuando el algoritmo on-line sirve completamente un bloque de una secuencia, el adversario elige una super-página p' de U' y la expande en un bloque de l pedidos constituidos por páginas de U correspondientes a p' . Este bloque de pedidos tiene la propiedad de que cuando el algoritmo on-line ve el pedido i -ésimo del bloque, la página pedida no se encuentra en la caché. El adversario siempre puede lograr esto pues la caché contiene k páginas, y el bloque se forma a partir de pedidos a $k+1$ páginas, existiendo siempre una que no se encuentra en la caché. La construcción de la instancia I continúa repitiendo este proceso hasta que el adversario decide terminar todas las secuencias.

Considerando los l pedidos de un bloque correspondiente a la super-página p' como un pedido a esta super-página, podemos ver las secuencias de super-páginas como una instancia I' para el problema $MTP(1, w)$. Del mismo modo podemos imaginar un algoritmo on-line A' para $MTP(1, w)$ que sirve un pedido a la super-página p' cada vez que el algoritmo A termina de servir un bloque correspondiente a p' .

Cada vez que el algoritmo A' sirve un pedido sin aparearlo con otro pedido el algoritmo A sirve un bloque completo de l pedidos. Por construcción del bloque, el algoritmo A incurre en l page-faults para servir este bloque. El algoritmo A no aparea ninguno de estos pedidos pues si así fuera el algoritmo A' también hubiese apareado el pedido servido.

El mismo razonamiento muestra que si el algoritmo A' aparea r pedidos, el algoritmo A puede aparear a lo sumo r bloques, incurriendo en por lo menos l page-faults para servir los lr pedidos. En consecuencia, por cada page-fault incurrido por el algoritmo A' , el algoritmo A incurre en l page-faults. Esto implica que:

$$C_A(I) \geq l C_{A'}(I')$$

El adversario puede servir la instancia I sirviendo juntos los bloques de l pedidos tal como el óptimo sirve los pedidos de la instancia I' . Por cada apareo de r pedidos realizado al servir la instancia I' , el adversario debe servir r bloques de l pedidos a $k+1$ páginas distintas. Cada bloque corresponde a un pedido de la instancia I' .

Eliendo l suficientemente grande, el adversario puede asegurarse una cota superior al costo óptimo de servir estos r bloques de l pedidos a $k+1$ páginas distintas. El lema 7.11 nos dice que si l es suficientemente grande es posible servir estos pedidos con a lo sumo

$\lceil \frac{e.l}{k+e/2} \rceil + k$ page-faults. El término k proviene del costo necesario para conseguir las condiciones iniciales supuestas en el lema, que exigían la caché cargada con k páginas de las $k+1$ existentes en los bloques.

Esta cota nos permite afirmar que por cada page-fault incurrido al servir la instancia I' en forma óptima se incurre a lo sumo $\lceil \frac{e.l}{k+e/2} \rceil + k$ page-faults en la instancia I . Esto implica que:

$$C_o(I) \leq (\lceil \frac{e.l}{k+e/2} \rceil + k) C_o(I') \leq (\frac{e.l}{k+e/2} + k+1) C_o(I')$$

Como por hipótesis d es una cota inferior para $MTP(1, w)$, para todo b el adversario puede armar una instancia I' tal que $C_{A'}(I') > d C_o(I') + b$. Entonces:

$$\begin{aligned} C_A(I) &\geq l C_{A'}(I') > l d C_o(I') + l b \geq l d \frac{C_o(I)}{\frac{e.l}{k+e/2} + k+1} + l b = \\ &= d \frac{C_o(I)}{\frac{e}{k+e/2} + \frac{k+1}{l}} + l b = d (k+e/2) \frac{C_o(I)}{e + \frac{(k+1)(k+e/2)}{l}} + l b = \\ &= d \frac{k+e/2}{e + \frac{(k+1)(k+e/2)}{l}} C_o(I) + l b = \\ &= \left(d \frac{k+e/2}{e+\varepsilon} \right) C_o(I) + l b. \text{ [Siendo } \varepsilon = \frac{(k+1)(k+e/2)}{l} \text{]} \end{aligned}$$

Si llamamos b' a $l b$ (notar que $l b$ puede ser visto una constante una vez elegido l) tenemos:

$$C_A(I) > \left(d \frac{k+e/2}{e+\varepsilon} \right) C_o(I) + b'$$

Como este resultado es válido para todo algoritmo A , tenemos que $\left(d \frac{k+e/2}{e+\varepsilon} \right)$ es una cota inferior para el problema $MTP(k, w)$. El valor de l es arbitrario mientras que $(k+1).(k+e/2)$ es una constante, por lo que eligiendo un l suficientemente grande se puede hacer ε tan chico como se quiera. Por lo tanto, $\left(d \frac{k+e/2}{e} \right)$ es una cota inferior para el problema $MTP(k, w)$. ■

8.4 Mejora de la cota inferior

En el capítulo 6 se vio la cota inferior $\frac{\lfloor \log w \rfloor}{2} + 1$ para el problema MTP(1,w). Usando este resultado y el lema 8.2 visto en la sección anterior podemos encontrar una cota inferior para el problema MTP(k,w).

Lema 8.3 : No existe un algoritmo on-line para el problema MTP(k,w) c -competitivo con $c < \left(\frac{\lfloor \log w \rfloor}{2} + 1 \right) \frac{k+e/2}{e}$.

Demostración:

El lema 6.8 nos dice que $\frac{\lfloor \log w \rfloor}{2} + 1$ es una cota inferior para la competitividad de un algoritmo on-line para el problema MTP(1,w).

El lema 8.2 nos permite generalizar esta cota inferior para el caso donde la caché es de tamaño k arbitrario.

Juntando ambos lemas obtenemos que $\left(\frac{\lfloor \log w \rfloor}{2} + 1 \right) \frac{k+e/2}{e}$ es una cota inferior para la competitividad de un algoritmo on-line para el problema MTP(k,w).
■

En el caso en que $k=1$ esta cota no es útil pues es menor que $\frac{\lfloor \log w \rfloor}{2} + 1$. Por lo tanto este lema provee una mejor cota sólo cuando $k \geq 2$.

9. Algoritmos Aleatorios

9.1 Introducción

En este capítulo veremos la extensión de la definición de competitividad para contemplar algoritmos on-line aleatorios. Mostraremos una cota inferior para el problema $MTP(1, w)$ cuando se tienen en cuenta dichos algoritmos. La cota encontrada es $1 + \frac{1}{w} - \frac{1}{w^2}$.

9.2 Competitividad y algoritmos aleatorios

En el trabajo de Manasse et al. [9] se extendió la definición de competitividad para incluir algoritmos on-line aleatorios. Estos algoritmos pueden usar valores aleatorios para decidir qué acciones tomar a lo largo de su ejecución. Por lo tanto, el costo sobre una instancia es una variable aleatoria, y la definición de competitividad debe adaptarse para reflejar este hecho.

Se dice que un algoritmo es c -competitivo si para toda instancia I el valor esperado del costo sobre la instancia I es menor o igual que c multiplicado por el costo óptimo off-line más una constante.

Formalmente, sea A un algoritmo on-line aleatorio, $E(C_A(I))$ el valor esperado del costo del algoritmo A sobre la instancia I y $C_o(I)$ el costo óptimo off-line.

El algoritmo A es c -competitivo si existe una constante b tal que, para toda instancia I del problema se tiene:

$$E(C_A(I)) \leq c C_o(I) + b$$

Claramente toda cota superior obtenida para un problema con algoritmos determinísticos es una cota superior considerando la nueva definición, pues en ese caso el costo no es una variable aleatoria y el valor esperado es directamente $C_A(I)$. Sin embargo, en general los algoritmos aleatorios logran obtener mejor performance que los determinísticos. Esto se debe a que el adversario no puede estar seguro de la acción tomada por el algoritmo y por ende es más difícil generar una “mala” instancia.

9.3 Cota inferior

Veremos a continuación una cota inferior para el problema MTP on-line aleatorio para el caso en que la caché es de tamaño uno.

Mostraremos una instancia tal que el valor esperado del costo es por lo menos $l(w + 1 - \frac{1}{w})$ mientras que el costo óptimo es lw , para todo entero positivo l . Esto nos

proporciona una cota inferior igual a $\frac{l(w + 1 - \frac{1}{w})}{lw} = 1 + \frac{1}{w} - \frac{1}{w^2}$. La cota inferior mostrada en esta tesis cuando no se consideran algoritmos probabilísticos es $\frac{\lfloor \log w \rfloor}{2} + 1$.

Lema 9.1: No existe algoritmo on-line aleatorio c -competitivo con $c < 1 + \frac{1}{w} - \frac{1}{w^2}$ para el problema MTP(1, w).

Demostración:

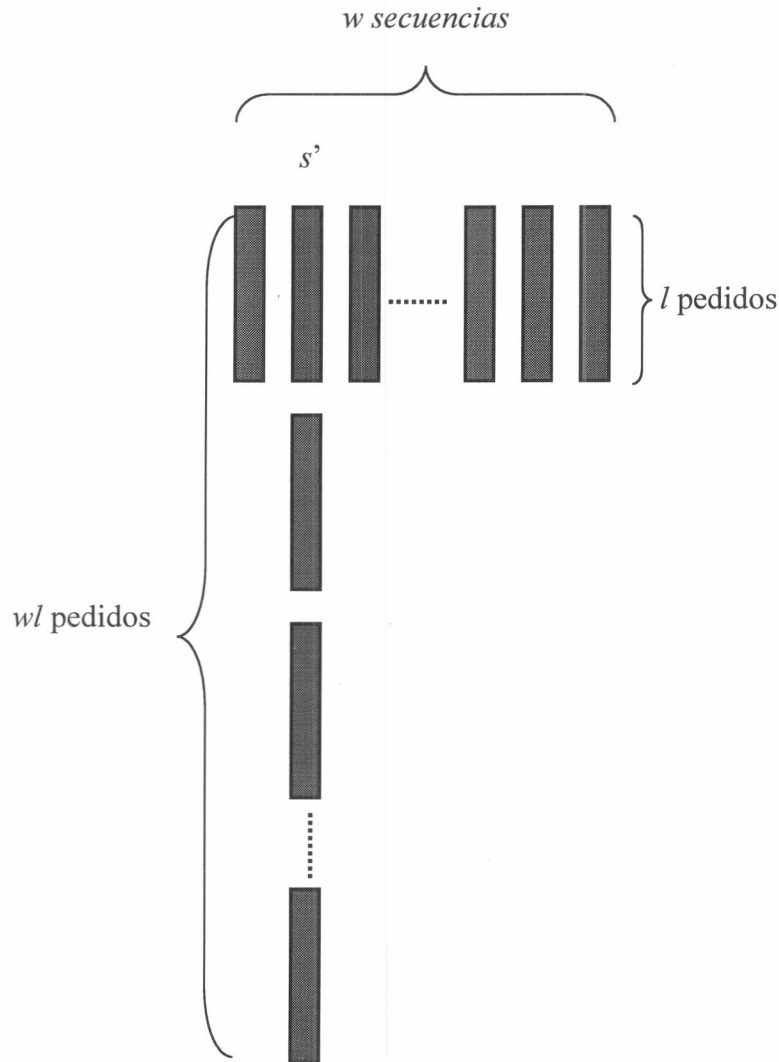
Sea A un algoritmo on-line aleatorio para el problema MTP(1, w) y l un número entero positivo arbitrario. Consideremos al algoritmo A frente a una instancia formada por w secuencias de l pedidos distintos cada una. El algoritmo A finaliza de servir completamente las w secuencias en cierto orden, en forma aleatoria.

Sea $P(s)$ la probabilidad de que la secuencia s sea servida completamente en último lugar. Llamemos s' la secuencia tal que $P(s')$ sea máximo. Por lo tanto, $P(s')$ tiene que ser mayor a $1/w$.

El adversario genera la instancia I de la siguiente manera:

- Armar las $w-1$ secuencias distintas de s' con l pedidos a páginas distintas.
- Armar la secuencia s' con l pedidos a páginas distintas seguido de los $(w-1)l$ pedidos a páginas de las otras secuencias.

El siguiente diagrama muestra un ejemplo de la instancia I



El costo óptimo off-line para esta instancia es lw , y se obtiene sirviendo la secuencia s' y junto con ella las $w-1$ secuencias restantes. Por lo tanto:

$$C_o(I) = lw$$

Sobre esta instancia, el algoritmo A termina la secuencia s' en último lugar con probabilidad al menos $1/w$. En esos casos obtiene un costo de $lw + l(w-1) = 2lw - l$.

Por otro lado, con probabilidad $\frac{w-1}{w}$ termina una secuencia distinta de s' en último lugar, obteniendo un costo de por lo menos lw .

En consecuencia, el valor esperado del algoritmo A sobre esta instancia es por lo menos:

$$E(C_A(I)) \geq \frac{1}{w} (2lw - l) + \frac{w-1}{w} lw = 2l - \frac{l}{w} + (w-1)l = l(w + 1 - \frac{1}{w})$$

Por lo tanto, obtenemos que

$$E(C_A(I)) \geq l \left(w + 1 - \frac{1}{w} \right) = \left(1 + \frac{1}{w} - \frac{1}{w^2} \right) l w = \left(1 + \frac{1}{w} - \frac{1}{w^2} \right) C_o(I)$$

■

10. Conclusiones

En esta tesis se realizó un análisis del problema MTP on-line desde diferentes ángulos. Se obtuvo una cota inferior para el caso en que tenemos una caché de tamaño uno. Esta cota mejora substancialmente la cota conocida anteriormente, pues esta última era menor a 2 para cualquier cantidad de secuencias. La nueva cota crece logarítmicamente con respecto a la cantidad de secuencias, mientras que la cota superior conocida crece linealmente. Por lo tanto, aún queda una brecha entre ambas cotas y campos abiertos para la investigación.

También se mostró cómo puede utilizarse esta cota (y cualquier otra) para obtener cotas inferiores cuando la caché es de tamaño mayor a uno. Utilizando este resultado se mejoró la cota inferior para el problema $MTP(k, w)$. En este caso también queda una brecha entre las cotas inferiores y superiores. Para la obtención de esta cota inferior se utilizó una expansión de un pedido en un bloque de l pedidos a $k+1$ páginas distintas. Creemos que es posible mejorar esta cota inferior si se consideran expansiones de más de $k+1$ páginas distintas, de modo de poder armar una estructura que el adversario pueda servir eficientemente.

Tanto la cota inferior como la superior crecen linealmente con el valor de k . Sin embargo, con respecto a w la cota inferior crece logarítmicamente y la superior linealmente. Por lo tanto, es importante investigar el comportamiento del problema para diferentes valores de w . Suponiendo una caché de tamaño uno y variando la cantidad de secuencias, se pueden obtener nuevas cotas inferiores. Estas cotas pueden generalizarse a otros tamaños de la caché usando los resultados de esta tesis.

Como la nueva cota inferior encontrada para el problema $MTP(1,2)$ no mejora la cota conocida anteriormente, se analizó este caso en particular, mejorando la cota inferior. Es importante determinar la mínima competitividad que puede tener un algoritmo on-line para este caso particular, pues este valor permitiría obtener cotas para el caso MTP en general.

En este trabajo se analizaron también problemas relacionados con el problema MTP. En el capítulo 5 se vio una variante del problema MTP que llamamos MTP Semi-Online, encontrando cotas inferiores y superiores para el caso $MTP(1,2,1)$. Queda abierto el problema de hallar cotas de competitividad para el caso general $MTP(k, w, v)$.

Asimismo, en el capítulo 7 se introdujo un problema off-line, cuya solución permitió obtener nuevos resultados para el problema $MTP(k, w)$. Este problema asume que la caché es de tamaño k y la cantidad de páginas distintas es $k + 1$. Sería interesante analizar el caso en que la cantidad de páginas distintas es $k + p$, para $p > 1$. El caso en que $p = w$ es de particular interés, pues parece posible formar secuencias de pedidos para las cuales el algoritmo debe fallar en todos los servicios.

La siguiente tabla compara las cotas encontradas con las conocidas anteriormente.

	Cotas Inferiores		Cotas Superiores
	Anterior	Actual	Actual
MTP(1,2)	1.5	1.788	2
MTP(1,w)	$2 - \frac{1}{w}$	$\frac{\lfloor \log w \rfloor}{2} + 1$	w
MTP(k,w)	$k + 1 - \frac{1}{w}$	$\left(\frac{\lfloor \log w \rfloor}{2} + 1 \right) \frac{k+e/2}{e}$	kw
MTP(1,w) con alg. on-line aleatorios	-	$1 + \frac{1}{w} - \frac{1}{w^2}$	2
MTP(1,2,1)	-	ϕ	2

11. Apéndice A

En este apéndice mostraremos que la función $f(\alpha) = (1-\alpha)^{1-1/\alpha} + e\alpha/2$ es menor que e en el intervalo $0 < \alpha < 1$. Este resultado fue necesario en el capítulo 7.

Primero probaremos algunos resultados que nos permitirán demostrar más fácilmente esta propiedad.

Lema A.1: $\lim_{\alpha \rightarrow 0^+} (1-\alpha)^{1-1/\alpha} = e$.

Demostración:

$$\begin{aligned}
 & \lim_{\alpha \rightarrow 0^+} (1-\alpha)^{1-1/\alpha} = [\text{Haciendo } n=1/\alpha] \\
 & = \lim_{n \rightarrow \infty} (1-1/n)^{1-n} = \\
 & = \lim_{n \rightarrow \infty} \left(\frac{n-1}{n} \right)^{1-n} = \\
 & = \lim_{n \rightarrow \infty} \left(\frac{n}{n-1} \right)^{n-1} = \\
 & = \lim_{n \rightarrow \infty} \left(\frac{n-1+1}{n-1} \right)^{n-1} = \\
 & = \lim_{n \rightarrow \infty} \left(1 + \frac{1}{n-1} \right)^{n-1} = \\
 & = \lim_{n \rightarrow \infty} \left(1 + \frac{1}{n} \right)^n = \\
 & = e.
 \end{aligned}$$

■

Lema A.2: Sea $g(\alpha) = 2\alpha \ln \alpha - (1-\alpha)^2$. Entonces $g(\alpha) < 0$ para $0 < \alpha < 1$.

Demostración:

El lema es evidente cuando notamos que $\ln \alpha$ es negativo cuando $0 < \alpha < 1$.

■

Lema A.3: Sea $g(\alpha) = e/2 + \left(\frac{(1-\alpha) \ln(1-\alpha) + \alpha}{\alpha^2} - 1 \right) (1-\alpha)^{-1/\alpha}$. Entonces $g(\alpha) < 0$ para $0 < \alpha < 1$.

Demostración:

Probaremos el lema en dos partes. Primero mostraremos que $\lim_{\alpha \rightarrow 0^+} g(\alpha) = 0$. Luego mostraremos que $g(\alpha)$ es decreciente en el intervalo $0 < \alpha < 1$. Como la función es decreciente y tiene $\lim_{\alpha \rightarrow 0^+}$ igual a 0, debe ser negativa en el intervalo indicado.

$$\begin{aligned}
 \text{A)} \quad \lim_{\alpha \rightarrow 0^+} g(\alpha) &= \\
 &= \lim_{\alpha \rightarrow 0^+} e/2 + \left(\frac{(1-\alpha) \ln(1-\alpha) + \alpha}{\alpha^2} - 1 \right) (1-\alpha)^{-1/\alpha} = \\
 &= e/2 + \lim_{\alpha \rightarrow 0^+} \left(\frac{(1-\alpha) \ln(1-\alpha) + \alpha}{\alpha^2} - 1 \right) \lim_{\alpha \rightarrow 0^+} (1-\alpha)^{-1/\alpha} = \\
 &= e/2 + \lim_{\alpha \rightarrow 0^+} \left(\frac{(1-\alpha) \ln(1-\alpha) + \alpha}{\alpha^2} - 1 \right) e = \\
 &= e \left(1/2 + \lim_{\alpha \rightarrow 0^+} \left(\frac{(1-\alpha) \ln(1-\alpha) + \alpha}{\alpha^2} - 1 \right) \right) = \\
 &= e \left(-1/2 + \lim_{\alpha \rightarrow 0^+} \frac{(1-\alpha) \ln(1-\alpha) + \alpha}{\alpha^2} \right) = \\
 &= e \left(-1/2 + \lim_{\alpha \rightarrow 0^+} \frac{(1-\alpha) \ln(1-\alpha) + \alpha}{\alpha^2} \right) = [\text{aplicando la regla de L'Hospital}] \\
 &= e \left(-1/2 + \lim_{\alpha \rightarrow 0^+} \frac{-\ln(1-\alpha)}{2\alpha} \right) = \\
 &= -e/2 \left(1 + \lim_{\alpha \rightarrow 0^+} \frac{\ln(1-\alpha)}{\alpha} \right) = [\text{aplicando la regla de L'Hospital}] \\
 &= -e/2 \left(1 + \lim_{\alpha \rightarrow 0^+} \frac{1}{\alpha-1} \right) = \\
 &= -e/2 (1 - 1) = \\
 &= -e/2 \cdot 0 = \\
 &= 0
 \end{aligned}$$

Por lo tanto, $\lim_{\alpha \rightarrow 0^+} g(\alpha) = 0$.

B) Para probar que $g(\alpha)$ es decreciente, mostraremos que su derivada es negativa en el intervalo $0 < \alpha < 1$.

La derivada de $g(\alpha)$ es $\left(\frac{(1-\alpha) \ln(1-\alpha)^2}{\alpha^4} - \frac{1}{\alpha^2} \right) (1-\alpha)^{-1/\alpha}$. Como $(1-\alpha)^{-1/\alpha} > 0$ en el intervalo, alcanza con probar que $\frac{(1-\alpha) \ln(1-\alpha)^2}{\alpha^4} - \frac{1}{\alpha^2} < 0$ en el intervalo. Entonces:

$$\frac{(1-\alpha) \ln(1-\alpha)^2}{\alpha^4} - \frac{1}{\alpha^2} < 0 \Leftrightarrow$$

$$\begin{aligned}
 &\Leftrightarrow \frac{(1-\alpha) \ln (1-\alpha)^2}{\alpha^2} - 1 < 0 \Leftrightarrow \\
 &\Leftrightarrow \frac{(1-\alpha) \ln (1-\alpha)^2}{\alpha^2} < 1 \Leftrightarrow \\
 &\Leftrightarrow (1-\alpha) \ln (1-\alpha)^2 < \alpha^2 \Leftrightarrow \\
 &\Leftrightarrow (1-\alpha) \ln (1-\alpha)^2 < \alpha^2 \Leftrightarrow [\text{haciendo } \beta=1-\alpha \Rightarrow 0<\beta<1] \\
 &\Leftrightarrow 2\beta \ln \beta < (1-\beta)^2 \Leftrightarrow \\
 &\Leftrightarrow 2\beta \ln \beta - (1-\beta)^2 < 0.
 \end{aligned}$$

Por el lema anterior $2\beta \ln \beta - (1-\beta)^2 < 0$ en el intervalo $0<\beta<1 \Rightarrow$
 $\frac{(1-\alpha) \ln (1-\alpha)^2}{\alpha^4} - \frac{1}{\alpha^2} < 0.$

■

Finalmente podemos demostrar el resultado buscado.

Lema A.4: Sea $f(\alpha) = (1-\alpha)^{1-1/\alpha} + e\alpha/2$. Entonces $f(\alpha) < e$ para $0<\alpha<1$.

Demostración:

Al igual que en el lema anterior, probaremos este lema en dos partes. Primero mostraremos que $\lim_{\alpha \rightarrow 0+} f(\alpha) = e$. Luego mostraremos que $f(\alpha)$ es decreciente en el intervalo $0<\alpha<1$.

$$\begin{aligned}
 \text{A)} \quad &\lim_{\alpha \rightarrow 0+} f(\alpha) = \\
 &= \lim_{\alpha \rightarrow 0+} (1-\alpha)^{-1/\alpha} + e\alpha/2 = \\
 &= \lim_{\alpha \rightarrow 0+} (1-\alpha)^{-1/\alpha} + \lim_{\alpha \rightarrow 0+} e\alpha/2 = \\
 &= \lim_{\alpha \rightarrow 0+} (1-\alpha)^{-1/\alpha} + 0 = [\text{Usando el lema A.1}] \\
 &= e.
 \end{aligned}$$

por lo tanto, $\lim_{\alpha \rightarrow 0+} f(\alpha) = e$.

B) Para probar que $f(\alpha)$ es decreciente, mostraremos que su derivada es negativa en el intervalo $0<\alpha<1$.

$$\text{La derivada de } f(\alpha) \text{ es } e/2 + \left(\frac{(1-\alpha) \ln (1-\alpha) + \alpha}{\alpha^2} - 1 \right) (1-\alpha)^{-1/\alpha}.$$

Por el lema A.3, esta función es negativa en el intervalo $0<\alpha<1$.

■

12. Referencias

- [1] E. Feuerstein. On-line paging of structured data and multi-threaded paging. Tesis de Doctorado, Università di Roma "La Sapienza", 1995.
- [2] E. Feuerstein, A. Strejilevich de Loma. On multi-threaded paging. Proc. Seventh Annual International Symposium on Algorithms and Computation (ISAAC'96), Volume 1178 of Lecture Notes in Computer Science, pp. 417-426. Springer - Verlag.
- [3] A. Fiat, R. Karp, M. Luby, L. McGeoch, D. Sleator, N. Young. Competitive paging algorithms. Journal of Algorithms, 12:685-699, 1991.
- [4] H. Alborzi, E. Torng, P. Uthaisombut, S. Wagner. The k -client Problem Proceedings of the eighth annual ACM-SIAM Symposium on discrete algorithms, New Orleans, Louisiana, pp. 73-82. 1997.
- [5] Y. Bartal, A. Fiat, H. Karloff, R. Vohra. New Algorithms for an Ancient Scheduling Problem. 24th Annual ACM STOC, may 1992
- [6] S. Ben-David, A. Borodin, R. Karp, G. Tardos, A. Widgerson. On the power of randomization in online algorithms. Technical Report TR-90-023, ICSI, June 1990.
- [7] A. Borodin, N. Linial, and M. Saks. An optimal online algorithm for metrical task systems. J. ACM 39, 4, 745-763. 1992.
- [8] R. Karp. On-line algorithms versus off-line algorithms: how much is it worth to know the future? Technical Report TR-92-044, ICSI, July 1992.
- [9] M. Manasse, L. McGeoch, D. Sleator. Competitive algorithms for server problems. Journal of Algorithms, 11(2):208-230, 1990.
- [10] D. Sleator and R. Tarjan. Amortized efficiency of list update and paging rules. Communications of ACM, 28:202-208, 1985.
- [11] P. Raghavan, M. Snir. Memory Versus Randomization in On-Line Algorithms. Research Report.
- [12] A. Fiat, A. Karlin. Randomized and multipointer paging with locality of reference. Proc. of 27th ACM Symposium on theory of computing. 1995
- [13] A. Borodin, S. Irani, P. Raghavan, B. Schieber. Competitive Paging with Locality of Reference. Proc. of 23rd ACM Symposium on theory of computing. 1991

- [14] J. Garay, I. Gopal, S. Kutten. Efficient On-Line Call Control Algorithms. *Journal of Algorithms*, 23:180-194, 1997.
- [15] B. Pittel, R. Weishaar. On-Line Coloring of Sparse Random Graphs and Random Trees. *Journal of Algorithms*, 23:195-205, 1997.
- [16] S. Irani, A. Karlin, S. Phillips. Strongly Competitive Algorithms for Paging with Locality of Reference
- [17] D. Hochbaum. Approximation algorithms for NP-hard problems. 1997, PWS Publishing Company, Boston, USA.
- [18] L. Belady. A study of replacement algorithms for virtual storage computers. *IBM Systems Journal*, 5:78-101, 1996.
- [19] L. Wang, T Jiang. On the complexity of multiple sequence alignment. *Journal of Computational Biology*, 1(4):337-348, 1994.