ALGORITMO LEXICOGRÁFICO PARALELIZABLE PARA LA BÚSQUEDA DE CAMINOS ÓPTIMOS

Maldonado E., Felipe Ciurlizza G., Augusto Morales M., Luis Bernardo

Resumen: En este trabajo se encara el desarrollo de un algoritmo que genere las permutaciones de n elementos, con el propósito de utilizarlo en procesos de búsqueda exhaustiva para la resolución de problemas de optimización combinatoria. Se presenta aquí un algoritmo secuencial que genera las permutaciones de n elementos en orden lexicográfico. El número total de intercambios necesarios para generar con el algoritmo propuesto las permutaciones de n elementos es 1,543n!, y al igual que el tiempo de CPU utilizado, es menor que los reportados con anterioridad. Cabe destacar además que la única información que utiliza para generar cada permutación es la permutación precedente, por lo que resulta fácilmente paralelizable bajo cualquier arquitectura SIMD. Se ha usado el algoritmo en la optimización de un proceso textil modelado como un agente viajero asimétrico y se pudieron encontrar secuencias de colores óptimas para el teñido de telas trabajando en una computadora personal. Este último hecho resulta trascendental porque el tipo de equipo necesario condiciona la aplicabilidad en el sector industrial.

Palabras clave: Algoritmo lexicográfico/ Caminos óptimos/ Optimización/ Paralelismo/ Permutaciones.

PARALLELIZABLE LEXICOGRAPHIC ALGORITHM IN THE SEARCH FOR OPTIMUM PATHS

Abstract: In this paper is considered the development of an algorithm that generates the permutations of n elements, with the purpose of using it in exhaustive search processes for the resolution of combinatorial optimization problems. A sequential algorithm that may generate the permutations of n elements in lexicographic order is presented. The total number of necessary interchanges for the generation of the permutations of n elements, together with the proposed algorithm, is 1,543n! and the CPU time used is less than those formerly reported. Besides, it is worth pointing out that the only information used to generate each permutation is the preceding one, reason why it is easily parallelizable under any SIMD architecture. The algorithm has been used in the optimization of a modeled textile process as an asymmetric travelling agent and optimal colour sequences could be found for the dyeing of the fabrics, working with a personal computer. This latest realization is very significant since the type of equipment required determines the applicability of the method in the industrial sector.

Keywords: Lexicographic Algorithm/ Optimization/ Optimum Paths/ Parallelism/ Permutation.

I. INTRODUCCIÓN

La generación de permutaciones ha sido, desde hace varias décadas, un problema importante en computación, tanto intrínsecamente, por la búsqueda de métodos que tratan de disminuir el tiempo de máquina, la memoria utilizada, etc., como por la gran cantidad de aplicaciones que tiene en ciencias e ingeniería.

El primer algoritmo computacional data de 1956 y es debido a Tomkins [1]; a partir de allí se ha desarrollado una variedad de algoritmos secuenciales que generan permutaciones con cambio mínimo [2], [3], en orden lexicográfico [2], [4], [5] e inclusive sin un orden especial [2].

Más recientemente se han publicado varios

Manuscrito finalizado el 06/05/98, recibido el 04/06/98, en su forma final (aceptado) el 05/02/99. El MSc Felipe Maldonado Etchegaray es Profesor Investigador en la Escuela Superior de Ingeniería Textil, Instituto Politécnico Nacional, Insurgentes Sur 3493, Edif. 27-503, México, DF 14020, Fax: +52 5 7296000 extensión 55192, e-mail: fmaldon@vmredipn.ipn.mx. El Dr. Augusto Ciurlizza Guizar es también Profesor Investigador en la misma Escuela del Instituto Politécnico Nacional, misma dirección y fax, e-mail: aguizar@vmredipn.ipn.mx. El Dr. Luis Bernardo Morales M. Es Investigador Titular en el Instituto de Investigaciones Matemáticas Aplicadas y en Sistemas, Universidad Nacional Autónoma de México, Apdo. Postal 70-221, México, DF 04510 México, Tel: +52 5 6223588, Fax: 6223596, e-mail: lbm@servidor.unam.mx.

algoritmos en paralelo [6]-[12], en arquitecturas SIMD [8], en diseños sistólicos [12] o con procesadores vectoriales [10]. Para la evaluación de estos algoritmos se consideran parámetros como costo computacional, modelo computacional y orden de la generación de las permutaciones.

Sin embargo, a pesar que la mayoría de los autores destacan la importancia que tiene la generación de permutaciones para resolver problemas combinatorios, no se encuentra en estos trabajos referencias explícitas a las posibilidades o limitaciones de su aplicabilidad.

En un problema típico de optimización combinatoria,

$$Mín_{\pi}\ z\ (\pi(1),\pi(2),...,\pi(n)), \qquad (1)$$

donde el objetivo z se debe optimizar sobre un conjunto de permutaciones p, es necesario que el algoritmo genere las permutaciones ordenadamente y que permita eliminar subconjuntos de permutaciones donde, por algún criterio K, se tiene la certidumbre de que el objetivo no alcanza su óptimo.

Los autores de este trabajo han desarrollado un algoritmo que genera las permutaciones en orden lexicográfico, utilizando solamente como información la permutación precedente; es eficiente en el tiempo de máquina, puede implantarse en paralelo en cualquier arquitectura SIMD y cuando es usado para resolver problemas del tipo (1) permite eliminar todas las permutaciones de una rama.

Cabe aclarar que se usa el término "lexicográfico", que se ha extendido de palabras a números y a otros símbolos, en sentido amplio. En análisis combinatorio es común usarlo para el ordenamiento de cualquier conjunto cuyos cambios sean equivalentes a los de las palabras en un diccionario, inclusive el orden inverso y sus complementos. Los autores utilizan el orden inverso porque es el tipo de ordenamiento que se ha usado en la literatura; y por lo tanto resulta más sencillo comparar el algoritmo con los que lo preceden[2], [4], [5].

En el ítem 1 se presenta un algoritmo secuencial que genera todas las permutaciones de n elementos en orden lexicográfico y es comparado con éxito con los algoritmos publicados más eficientes. En el ítem 2 se presenta un análisis cuantitativo del desempeño del algoritmo al ser usado en una búsqueda exhaustiva. El ítem 3 está dedicado a un diseño de paralelización del algoritmo en un proceso de búsqueda.

II. DESARROLLO

1. Algoritmo secuencial

La mayoría de los algoritmos que generan las permutaciones de n elementos en orden lexicográfico utilizan un vector auxiliar, denominado "firma", en forma explícita o implícita [2]. Aunque la cantidad de memoria utilizada por el vector es despreciable, la generación de la "firma" y su lectura para la determinación de cada permutación, conllevan el uso de un tiempo considerable.

Sin embargo, cuando los elementos a permutar son alfanuméricos el vector auxiliar no es necesario.

El algoritmo que se ha desarrollado utiliza las siguientes características de las permutaciones ordenadas en forma lexicográfica:

a) Si se asigna el número 1 a la primera permutación, {1,2,...,n}, todas las permutaciones pares se obtienen de la anterior intercambiando los dos primeros elementos.
   Así, si la permutación k-ésima (k, impar) de n elementos es a1, a2, a3, ..., an, entonces, la permutación k+1-ésima se obtiene intercambiando a2 con a1, es decir, a2, a1, a3 ..., an.
b) Las permutaciones impares se obtienen intercambiando el primer elemento aj, desde a3 hasta an, por el primero, de su izquierda que sea menor a él y ordenando de menor a mayor los elementos de a1 a aj-1. Cuando se llega a an sin haber encontrado ningún elemento menor a su izquierda, el proceso de generación de las permutaciones ha concluido.

Nótese que para un conjunto de números {1,2,...,n}, el término lexicográfico, en sentido estricto, se refiere al ordenamiento de las n! permutaciones de menor a mayor. Entonces, dada la permutación k-ésima, es evidente que se obtiene la siguiente buscando de derecha a izquierda, el primer número que tenga a su derecha uno mayor a él y reordenando lexicográficamente el subconjunto. Si los números se escribieran en orden inverso {n,n-1,...1}, habría que invertir el sentido de la búsqueda; de igual manera que si se usan los complementos a n de los elementos.

Si se toma, por ejemplo, la permutación 42135 (permutación decimosegunda de los cinco elementos 1, 2, 3, 4, 5), la siguiente permutación se obtiene: a)intercambiando el elemento a4, porque es el primero, de izquierda a derecha, que tiene uno menor a él -el a2-, para obtener el arreglo intermedio 43125; y b)ordenando de menor a mayor los primeros 3 (4-1) elementos. Para ello se intercambia el elemento a1 por a3 -los primeros parte entera de j/2 elementos. La permutación número 13 es entonces 13425.

Formalmente, un pseudo código del algoritmo que se ha desarrollado tiene la estructura presentada a continuación:

```
Comenzar
    Escoger un valor de n
    Generar la primera permutación
        Repetir
            Intercambiar los dos primeros elementos
```

Buscar ak, entre a3 y an, con algún elemento menor a su izquierda.
Si existe, intercambiarlos. En caso contrario p es verdadero
Ordenar de menor a mayor el subconjunto {a1, a2, ..., ak-1}
Mientras el criterio p de parada sea falso
Fin

Codificando en C++ el algoritmo desarrollado, el de Phillips[4] y el de Ord-Smith[2] y corridos en una microcomputadora 586 a 133 MHz, se obtuvieron los tiempos consignados en la Tabla I.

Tabla I. Cuadro comparativo de los tiempos empleadospor tres algoritmos secuenciales.

Algoritmo	*n*=7	*n*=10 seg	*n*=13
Desarrollado	0,0021	1,37	2461
Phillips	0,0022	1,43	2563
Ord-Smith	0,0023	1,52	2725

El número total de intercambios necesarios para generar, con el algoritmo propuesto, las permutaciones de n elementos es 1.543n!. Este número se puede obtener con el siguiente análisis:

i) Para obtener las permutaciones pares, se efectúa un intercambio por cada una, o sea un total de 0,5n! intercambios.
ii) Para obtener las permutaciones impares se efectúan dos procesos. Un intercambio del elemento aj por el primero que sea menor a su izquierda y un ordenamiento. O sea, nuevamente, un total de 0,5n! intercambios más los intercambios necesarios para el reordenamiento de menor a mayor de j-1 elementos.

Designando con Pj-1 a la cantidad de intercambios realizados por reordenamientos cuando se permutan j-1 elementos, al permutar j elementos se efectuarán j*Pj-1 + (j-1)*|(j-1)/2| intercambios.

El último sumando se debe a que al permutar j elementos, el j-ésimo elemento es intercambiado una vez con cada uno de los j-1 elementos que le anteceden y en cada uno de esos casos, al reordenar se producen |(j-1)/2| intercambios.
Se obtiene así la relación de recurrencia

$$P_j = j*P_{j-1} + (j-1)*|(j-1)/2| \qquad (2)$$

con P3 = 2.

Entonces,

P4 = 4*P3 + 3*1 = 4*2 + 3
P5 = 5*P4 + 4*2 = 5*(4*P3 + 3) + 8
P6 = 6*P5 + 5*2 = 6*(5*(4*P3 + 3) + 8) + 10
...
...

Al sumar la serie para j = 10, se obtiene el valor 0,54308n!, y puede mostrarse que la serie converge a ese valor.

Los términos posteriores al décimo, (j=10), son menores que 1/2(j+1). Si se suma la serie $1/2^{10}*\Sigma 1/2^j$ (desde j=1 hasta ∞) se obtiene como cota el valor 0,0001, c,q,q.d.

2. Aplicaciones

Se ha utilizado el algoritmo desarrollado para el proceso de búsqueda del camino de mínimo costo en un problema industrial modelado como el agente viajero asimétrico.

En el teñido de telas se debe escoger una secuencia de colores tal que los residuos de los colorantes que permanecen en el equipo utilizado y lo contaminan, afecten en la menor medida posible los colores subsiguientes. Designando por Dij a las diferencia (Cj' - Cj), entre un color requerido Cj (en el sistema Lab, de la Comisión Internacional de Iluminación) y el color Cj' que resultaría al teñir después del color Ci; se obtiene la matriz D de costos, obviamente asimétrica. Una matriz típica, para 10 colores, se presenta en la Tabla II.

Tabla II. Matriz de las diferencias de color causadas por teñidos consecutivos

MATRIZ (Δ_E) DE DIFERENCIAS DE COLOR									
0,00	0,21	0,07	0,12	0,07	0,20	1,28	0,22	0,05	0,22
5,91	0,00	0,18	0,42	0,38	1,31	7,22	1,18	0,23	1,39
15,03	1,98	0,00	0,62	0,34	1,15	6,50	0,99	0,16	1,22
20,12	3,95	0,31	0,00	0,08	0,47	2,34	1,25	0,08	0,38
42,84	16,71	1,89	0,82	0,00	0,60	3,32	7,42	0,55	2,07
46,33	28,01	11,66	11,69	5,78	0,00	2,84	8,04	0,65	3,26
21,38	14,36	7,41	8,33	4,63	0,17	0,00	0,89	0,06	0,46
14,69	11,49	6,59	7,72	4,95	0,48	1,31	0,00	0,12	0,44
39,28	32,15	19,94	21,00	13,19	1,13	4,77	3,10	0,00	1,29
16,86	8,28	3,80	4,38	2,24	0,22	1,00	0,82	0,05	0,00

El procedimiento utilizado consiste en sumar, para cada permutación generada, el costo de los caminos parciales en el orden de menor a mayor velocidad de cambio e ir controlando si se supera el menor valor Q, obtenido hasta ese momento. Si al sumar k caminos parciales se supera la cota, entonces se genera la última permutación de esa rama ordenado de mayor a menor los n-k elementos restantes. Dado que, como se explicó anteriormente, el algoritmo desarrollado utiliza cada permutación para generar la siguiente, el proceso de búsqueda continúa desde la permutación subsiguiente a la generada por ordenamiento.

El tiempo utilizado para encontrar el camino óptimo, {12354(10)8769}, con costo 4,88, en la misma computadora 586 de 133 MHz, fue de 2,57 seg.

Si bien es cierto que los métodos exhaustivos de

búsqueda, tales como branch and bound o programación dinámica, sólo pueden usarse, por si mismo, para resolver instancias de pequeño tamaño, resulta de trascendental importancia en combinación con métodos heurísticos tanto en la determinación de cotas iniciales como en la intensificación en la vecindad de óptimos locales.

3. Paralelización

Se presenta aquí el procedimiento de paralelización para una computadora de memoria compartida:

a) Se siembra en la memoria la permutación inicial, {1, 2, ..., n} y una cota inicial Q.
b) El primer procesador lee en la memoria la cota Q y la permutación "semilla", {1, 2, ..., n}, devuelve a la memoria la permutación k!+1 siguiente, {1, 2, ..., k-2, k, k-1, ..., n} y realiza su tarea de búsqueda de la mejor permutación con el procedimiento explicado en el parágrafo anterior.
c) El segundo procesador lee en la memoria la cota Q y la permutación "semilla", {1, 2, ..., k-2, k, k-1, ..., n}, escribe en la memoria la permutación k!+1 siguiente y realiza su tarea de búsqueda. De la misma manera actúan los siguientes procesadores.
d) Cada vez que un procesador termina su trabajo de búsqueda, lee en la memoria la permutación "semilla" existente en ese momento. Si algún procesador encuentra una cota menor, Q', la siembra en la memoria.
e) Una bandera impide que dos procesadores lean la misma permutación; cuando uno comenzó a leer, otro no puede hacerlo. Se debe establecer una jerarquía entre los procesadores para evitar que dos puedan leer simultáneamente.

Es evidente que la estructura implantada evita que haya acciones superpuestas de dos o más procesadores al "cortar" durante el proceso de búsqueda. Si el procesador i-ésimo está realizando el proceso de búsqueda y corte (Figura 1) no puede existir ningún otro procesador j que realice la misma suma.

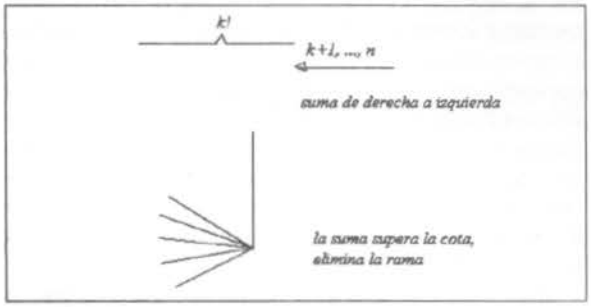

Figura 1. Representación esquemática de la eliminación de una rama en el proceso de búsqueda de un procesador i

III. CONCLUSIONES

1. El algoritmo secuencial desarrollado genera las permutaciones de n elementos en orden lexicográfico con una cantidad de intercambios y en menor tiempo que los reportados con anterioridad.
2. Dado que la única información que utiliza para generar cada permutación es la permutación precedente, avanza o comienza la generación desde cualquier permutación sin cálculos adicionales.
3. La paralelización en cualquier estructura SIMD es inmediata, lo que lo hace útil en los procesos de búsqueda.

IV. REFERENCIAS

1) Tomkins, C., "Machine attacks on problems whose variables are Permutations", Proc. 6th. Symp. Maths., Mc. Graw Hill, (1956).
2) Ord-Smith, R.J., "Generation of permutation sequence: Part 1", Comput. J., 13, 3 (1970).
3) Jonhson, S.M., "Generation of permutations by adjacent transposition ", Math. Comput., 17, 83, (1963).
4) Phillips, J.P., "Permutation of the elements of a vector in lexicographic order (Algorithm 28)", Comput. J., 10 (1967).
5) Shen, M.K., "Generation of permutations in lexicographical order (Algorithm 202)", Comput. J., 9, (1963).
6) Akl, S.G., "Adaptative and optimal parallel algorithms for enumerating permutations and combinations", Computer J., 30, (1987).
7) Chen, G.H. and Chern, M.S., "Parallel generation of permutations and combinations", BIT, 26, (1986).
8) Gupta, P. and Bhattacharjee, G.P., "Parallel generation of permutations", Comput. J., 26, (1963).
9) Lin, C.J., "Parallel generation of permutations on systolic arrays", Parallel Comput., 15 (1990).
10) Mor, M. and Fraenkel, A.S., "Permutation generation on vector processors", Comput. J., 25, (1982).
11) Tsay, J.C. and Lee, W.P., "An optimal parallel algorithm for generating permutations in minimal change order", Parallel C., 20, (1994).
12) Lee, W.P and Tsay, J.C., "A systolic design for generating permutations in lexicographic order", Parallel C., 20 (1994).