Pontificia Universidad Javeriana

Ensayo: Algoritmos De Aproximación

Juan Sebastián Prado Valero

En la vida cotidiana de las personas es común encontrarse con problemas que a simple vista puede no ser difíciles calcular la solución a cada uno de estos, pero usualmente encontrar la mejor solución o la óptima es bastante complicado y suele tener bastantes complicaciones y puede consumir bastantes recursos como el tiempo. Para la solución de estos problemas se pueden desarrollar o realizar algoritmos que intentan hallar una respuesta absolutamente correcta en un tiempo dado por una expresión polinomial y que pueda correr en una maquina secuencial (siguiendo la tesis extendida de Church – Turing), pero en la mayoría de casos no es posible garantizar que exista un algoritmo que cumpla con estas expectativas, por lo que usualmente pueden fallar en una o varias de estas, estos problemas los denominamos difíciles. [1]

Nosotros como Ingenieros de Sistemas podemos llegar a tener clientes que requieran una solución para algún problema difícil y al momento de buscar o realizar un algoritmo que pueda dar solución a este problema hay que tener en cuenta la opinión y las prioridades que tenga nuestro cliente. Este ensayo se va a enfocar en los clientes que priorizan un algoritmo bueno frente a una solución completamente correcta. En ese sentido, habría que buscar una alternativa al algoritmo requerido sacrificando una solución que siempre va a ser optima, siendo así que se sigan una serie de pasos para calcular una solución que sea cercana a la óptima. Lo anterior se puede lograr mediante la utilización de un algoritmo voraz.

Un algoritmo es voraz porque siempre escoge el mejor candidato para formar parte de la solución: aquel que tenga mejor valor de la función objetivo, lo que constituye el cumplimiento de cierto criterio goloso de selección. Además es miope porque esta elección es única e inmodificable dado que no analiza más allá los efectos de haber seleccionado un elemento como parte de la solución. No deshacen una selección ya realizada: una vez incorporado un elemento a la solución permanece hasta el final y cada vez que un candidato es rechazado, lo es permanentemente. [2]

Un ejemplo con un problema de la vida real es el de la programación de tareas a maquinas o task scheduling, el cual consite en que:

Dadas M máquinas (consideradas procesadores) y N tareas con Tij unidades de tiempo de duración de cada tarea i-ésima ejecutada en la máquina j-ésima. Se desea programar las N tareas en las M máquinas, procurando el orden de ejecución más apropiado, cumpliendo determinadas condiciones que satisfagan la optimalidad de la solución requerida para el problema. [3]

Una variante de este problema es cuando las tareas son dependientes y las máquinas son diferentes, en esta variante cada tarea presenta una lista de tareas que la preceden y para ser ejecutada deben esperar el procesamiento de dicha lista en su totalidad. A esta situación hay que agregarle la característica de heterogeneidad de las máquinas: cada tarea demora

tiempos distintos de ejecución en cada máquina. El objetivo será minimizar el tiempo acumulado de ejecución de las máquinas, conocido en la literatura como makespan. [4]

Al observar el estado del arte del problema, vemos que tanto su aplicación práctica de forma directa en la industria como su importancia académica, al ser un problema NP-difícil. [3]

Como se mencionó anteriormente, al ser un problema difícil, no existen métodos exactos para resolver el problema; por lo que se puede realizar una especie de algoritmo heurístico que busque una solución cercana a la óptima. Este problema puede ser representado mediante una asignación de variables binarias, en donde la función objetivo es una sumatoria de multiplicaciones de los tiempos que gastaría cada maquina en cada una de las tareas multiplicados con una correspondiente variable binaria que toma valores de 1 si se asigna la tarea a la maquina o 0 si no se asigna.

Para el planteamiento del algoritmo voraz se supone que se tiene una instancia de trabajo completa que incluya lo siguiente: cantidad de tareas y máquinas (N y M respectivamente), matriz de tiempos de ejecución T y lista de predecesoras por tarea. Además se considera que en el lote hay al menos una tarea sin predecesoras que se convertirá en la inicial, así como no existen referencias circulares entre las predecesoras de las tareas que impidan su correcta programación. [3] La estructura de datos que se utiliza es:

N: número de tareas $J_1, J_2, ..., J_N$
M: número de máquinas $M_1, M_2, ..., M_M$
Matriz T: [T_{ij}] MxN de tiempos de procesamiento, donde cada entrada representa el tiempo que se demora la máquina j-ésima en ejecutar la tarea i-ésima.
Vector A: [A_i] de tiempos de procesamiento acumulado, donde cada entrada A_i es el tiempo de trabajo acumulado de la máquina M_i.
P_k : Conjunto de tareas predecesoras de la tarea J_k.
Vector U: [U_k] de tiempos de finalización de cada tarea J_k.
Vector V: [V_k] de tiempos de finalización de las tareas predecesoras de J_k, donde se cumple que $V_k = \max\{U_r\}, J_r \in P_k$.
S_i : Conjunto de tareas asignadas a la máquina M_i.
E: Conjunto de tareas programadas.
C: Conjunto de tareas candidatas a ser programadas.

Figura 1. Estructura de datos [3]

Las consideraciones y/o restricciones que se toman en el algoritmo son: Como criterio de selección de una tarea se selecciona la que menos tiempo de ejecución posee y para seleccionar una máquina se selecciona la que ejecute la tarea en el menor tiempo. Las tareas que ya se hayan asignado a alguna máquina, no puede volver a ser asignada. Las maquinas no pueden tener mas de una tarea asignada. Además las tareas que tengan predecesoras no pueden ser ejecutadas hasta que las predecesoras sean ejecutadas completamente.

En el caso particular de las tareas cuyas predecesoras ya han sido ejecutadas en su totalidad seguiremos así: Como la tarea J_i se debe ejecutar después de todas sus predecesoras, debemos ubicar la predecesora que finaliza más tarde (termina de ser ejecutada en el mayor tiempo). La mejor máquina M_k se elegirá de entre los M valores posibles: será aquella que minimice el valor de la suma de la entrada de la matriz de tiempos de ejecución T_{ij} con el máximo de entre la entrada correspondiente al vector de tiempos acumulados respectiva A_k y al tiempo de finalización de la última de las predecesoras de: J_i:V_k. A la mejor máquina le asignamos la tarea J, actualizando al vector A en la entrada correspondiente a M_k. Se repite el proceso para el resto de tareas. [3]

Finalmente el mayor tiempo o entrada de la lista A será el makespan.

El algoritmo sería el siguiente:

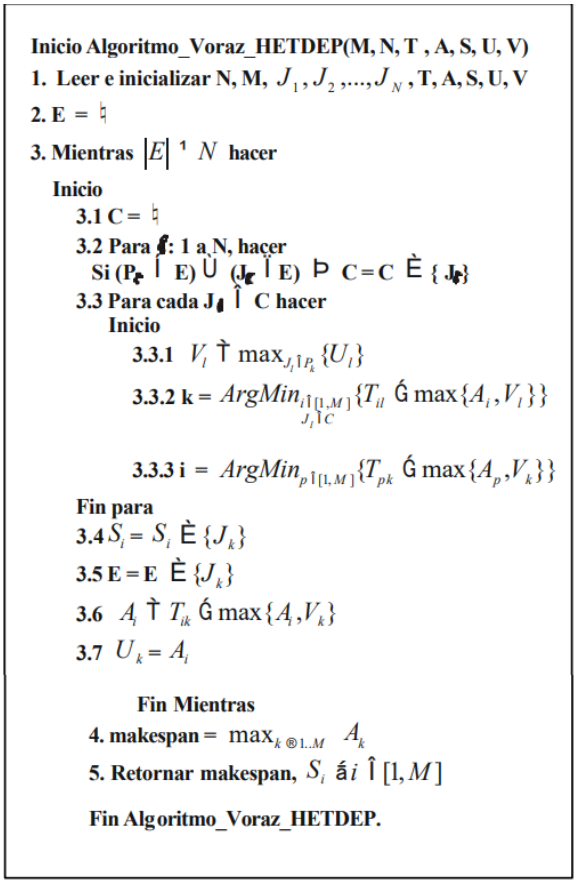

Figura 2. Algoritmo Voraz [3]

Para mayor información, revisar las referencias.

Referencias:

[1] Apuntes de Análisis de Algoritmos en clase con Danilo Castro, 2019

[2] Cormen T.; Leiserson Ch.; Rivest R. Introduction to Algorithms, MIT Press, Editorial McGraw Hill (2001)

[3] Tupia, M., & Mauricio, D. (2004). Un Algoritmo Voraz Para Resolver El Problema De La Programación De Tareas Dependientes En Máquinas Diferentes. Rev. Investig. Sist. Inform, Universidad Nacional Mayor de San Marcos, 1(1), 9–18.

[4] Campello R., Maculan N. Algoritmos e Heurísticas Desenvolvimiento e avaliaçao de performance. Apolo Nacional Editores, Brasil (1992).