UNIVERSIDAD DE SEVILLA

FACULTAD DE MATEMATICAS

" Problemas de Secuenciación con fechas de entrega "

Memoria dirigida por:

Don Rafael Infante Macías

José Luis Pino Mejías

INDICE

INTRODUCCION

Nota histórica

Los problemas de secuenciación-asignación ("scheduling") tienen su origen en las situaciones en que se requiere ordenar un conjunto de recursos, que son precisos para realizar una serie de actividades, en el tiempo.

Durante siglos la humanidad ha empleado métodos informales de secuenciación en las más variadas situaciones de la vida cotidiana. Sin embargo, las primeras técnicas formales tienen su origen durante la Primera Guerra Mundial con la aparición de los diagramas de Gantt. Estos gráficos de barras fueron ideados como herramienta para el tratamiento de los problemas originados por la distribución de las cargas en los barcos aliados.

Los primeros modelos matemáticos comienzan a aparecer sobre la mitad de la década de los cincuenta, pudiéndose considerar como pionero el trabajo de Johnson (1954).
A partir de este momento aumenta rápidamente el interés hacia este tipo de problemas,especialmente en las áreas de la Investigación Operativa, Ingeniería Industrial e Informática,obteniéndose una ingente cantidad de resultados, de gran interés,no solo desde un punto de vista teórico,sino por su aplicabilidad en campos tan dispares como puedan ser los Sistemas Industriales,Hospitalarios,Educativos,Urbanos,etc.

Este desarrollo no hubiera sido posible sin el de la Informática,tanto en su vertiente de "hardware" como en la de "software",baste citar,a modo de ejemplo,la importancia del empleo de las ideas que constituyen la base de la Programación Estructurada en el diseño y análisis de los algoritmos.
A su vez,la Teoría de la Secuenciación tiene uno de sus principales campos de aplicación en los sistemas de ordenadores, siendo en la actualidad uno de los principales puentes entre la Investigación Operativa y la Informática.

Hay que destacar,por último,la gran influencia que sobre el desarrollo teórico de las técnicas de secuenciación-asignación ha ejercido la Teoría de la Complejidad como consecuencia de los grandes avances de ésta desde el principio de los años setenta.

Objetivos

El objeto de la presente memoria es el estudio de algunos problemas de secuenciación-asignación cuando existen fechas de entrega.

La hemos estructurado en tres capítulos. En el primero de ellos se introducen los elementos fundamentales y se exponen las hipótesis que originan el modelo básico de problema de secuenciación-asignación, aportando la posibilidad de existencia de recursos adicionales distintos de las maquinas. El análisis de las situaciones en que algunas de las hipótesis son relajadas nos permite construir un esquema de clasificación que generaliza los hasta ahora desarrollados. Dentro de este capítulo damos una caracterización de las secuencias factibles utilizando el concepto de grafo disyuntivo.

El segundo capítulo presenta un compendio de los conceptos y resultados pertenecientes al campo de la Teoría de la Complejidad, que se precisan como bagaje para poder abordar el estudio de la "dificultad" de los problemas de secuenciación-asignación.

En el último capítulo primeramente se estudia la inviabilidad del método de enumeración completa , y se desarrolla una formulación por Programación Lineal Entera , analizandose su aplicabilidad.A continuación,divididimos el capítulo en dos apartados atendiendo, en principio, al tipo de criterio (minimización del máximo costo o del costo total) . Aunque podemos considerar que la característica fundamental que nos permite hacer la separación es la de la existencia

o no de "buenos" algoritmos para la resolución de los problemas. En el primer apartado,una vez expuestos los más potentes resultados conocidos, los puntos de mas interés son los relativos al caso de existir tiempos de procesamiento o disponibilidad de recursos variables. En el segundo se establece el carácter fuertemente NP-difícil del problema lo que puede tomarse como una justificación del empleo de técnicas enumerativas. Como principal resultado tenemos un procedimiento de resolución para el caso de recursos variables, que se basa en una formulación por Programación Dinámica que explota la existencia de relaciones de precedencia entre los trabajos, ya sean éstas debidas a las características tecnológicas del problema o generadas por condiciones de dominancia. Con nuestro método logramos una disminución de las necesidades de almacenamiento respecto de los procedimientos existentes.

De entre las etapas a dar para la resolución de un problema en el campo de la Investigación Operativa, solía relegarse a un segundo plano la de la implementación de los algoritmos, no teniéndose en cuenta la influencia que sobre la efectividad de los mismos tiene la implementación adoptada. En nuestro trabajo hemos intentado expresar los algoritmos de la forma más próxima posible a la utilizada en su implementación en el ordenador, limitándose solo en algunos casos esta cercanía por motivos de inteligibilidad. En el apéndice se introduce y justifica el lenguaje empleado.

CAPITULO I

Los modelos de secuenciación-asignación aparecen al estudiar aquellas situaciones en que se requiere procesar un conjunto dado de "tareas" utilizando unos "recursos",estando determinado el costo de la operación por el orden de procesamiento de las tareas.

Las características cuantitativas y cualitativas de los recursos deben ser conocidas.Las tareas quedan descritas por la cantidad de recursos que requieren para ser procesadas,el momento en que están disponibles, la duración del proceso y la fecha en que deben haber sido completadas.

En la mayoría de los modelos estudiados estos recursos están constituidos únicamente por un conjunto de elementos denominados <<máquinas>> o <<procesadores>>. En el caso más general,existen además un conjunto adicional de <<recursos>> un subconjunto de los cuales (posiblemente vacío) es requerido por cada tarea para poder ser procesada.

A las tareas se les denomina <<trabajos>> pudiendo estar constituidos por una o más <<operaciones>>.

A cada ordenación de los trabajos para su procesamiento le corresponde un costo. El objetivo en los problemas de secuenciación-asignación es obtener aquella ordenación a la que corresponda un coste mínimo.

Es conveniente desde el principio establecer la diferencia entre <<secuencia>> ("sequence") y <<secuencia-asignada>> o <<plan de trabajo>> ("schedule").Una secuencia factible corresponde a una simple determinación del orden en que deben ejecutarse las operaciones,mientras que una secuencia-asignada se obtiene al especificar los tiempos de comienzo y finalización de cada operación,por tanto a cada secuencia le corresponden infinitas secuencias-asignadas. Sin embargo,si

trabajamos en un ambiente completamente determinístico los términos de secuenciación-asignación y secuenciación pueden ser, en general,utilizados indistintamente,ya que si las características de los trabajos son conocidas a priori y un trabajo es comenzado tan pronto como sea posible, solo debe considerarse una asignación para cada secuencia.Por otro lado, en el caso estocástico debemos limitarnos a dar reglas para ordenar los trabajos, puesto que no podrán ser establecidas a priori las fechas de comienzo y finalización de las operaciones.

Por lo anterior, a lo largo de la presente memoria utilizaremos los términos Teoría de la Secuenciación y secuencia, siempre que no haya lugar a confusión.

Por último, debemos indicar que suponemos que todas las constantes y variables que intervienen en los problemas de secuenciación son enteras salvo que expresamente se señale lo contrario.

I.1 Notación y Definiciones

Mediante O denotamos al conjunto de todas las operaciones, O puede ser descompuesto en subconjuntos disjuntos de dos formas:

$$O = J_1 \cup J_2 \cup \ldots \cup J_n$$

$$O = M_1 \cup M_2 \cup \ldots \cup M_m$$

donde $J = \{ J_1, \ldots, J_n \}$ es el conjunto de los trabajos y

$M = \{ M_1, \ldots, M_m \}$ es el conjunto de las máquinas ,

consideramos además el conjunto de recursos

$$R = \{ R_1, \ldots, R_s \}$$

la cantidad máxima de recursos del tipo h disponible en cada instante viene dada por q_h ,siendo $\bar{q}_h$ la cantidad total de recursos del tipo h.

Las dos razones fundamentales para diferenciar entre recursos y máquinas son:

- Las máquinas constituyen un recurso forzósamente compartido por todos los trabajos.
- Las máquinas solo pueden procesar, en cada instante, como máximo una operación.

Suponemos que las operaciones que constituyen cada trabajo J_i están linealmente ordenadas por una relación " $<$ " de forma que

$$O_{i1} < O_{i2} < \ldots < O_{in_i}$$

donde n_i es el número de operaciones del trabajo J_i.

Si O_{ir} , $O_{is} \in J_i$ y $r < s$ decimos que O_{ir} precede a O_{is} ,si $s = r+1$ escribimos $O_{ir} << O_{is}$ y se dice que O_{ir} es predecesora inmediata de O_{is}.

Podemos renumerar la operación O_{ij} mediante O_u con

$$u = \sum_{K=1}^{J-1} n_K + i$$

y establecer una correspondencia

$$\# : O \longrightarrow \{ 1, \ldots , m \}$$

de forma que $\#(O_u)$ nos da el índice de la máquina en que se procesa O_u.

Existen diversos modos de representar los datos. Así, la representación más simple y empleada de los datos que determinan un modelo de secuenciación la constituyen los diagramas de Gantt.

Si cada trabajo y máquina tienen en común, a lo más, una operación ($\#(O_{iK})=K$) podemos recurrir a tres matrices, una T de orden $n \times m$ $T= [t_{iK}]$ donde t_{iK} es el tiempo de procesamiento del trabajo J_i en la máquina M_K, una matriz $n \times s$ $Q = [q_{ih}]$ siendo q_{ih} la cantidad de recurso R_h que requiere J_i para ser procesado, lógicamente deberá cumplirse que

$$\sum_{i=1}^{n} q_{ih} \leq \bar{q}_h \quad ,$$

y otra matriz $n \times m$ $S=[s_i(K)]$ que indica el orden de las operaciones de J_i en las máquinas por medio de la permutación s_i de los índices $\{1,\ldots,m\}$, de forma que

$$O_{is_i(1)} < O_{is_i(2)} < \ldots < O_{is_i(m)} \quad .$$

Sin embargo, es más útil trabajar con la representación basada en el concepto de grafo disyuntivo (Roy & Sussmann, 1964; Gorenstein,1972).

Representamos el problema de secuenciación mediante un grafo $G = (V,C,D)$ con:

- $V \subset \mathbb{N} \cup \{*\}$ conjunto de vértices que representan las operaciones, incluyendo dos ficticias, 0 y $\{*\}$, correspondientes al comienzo y el final, respectivamente.

Si denotamos

$$N_i = \sum_{j=1}^{i} n_j$$

el conjunto de los vértices que corresponden a operaciones iniciales se podrá representar

$$A_o = \{ r / r = N_i + 1 , i = 0,\ldots,n-1 \}$$

y el de los finales por

$$B_* = \{ r / r = N_i , i = 1,\ldots,n \}$$

- $C \subset V \times V$ conjunto de los arcos conjuntivos, cuyos sentidos representan el orden de procesamiento de las operaciones.

$(r,s) \in C$ si y solo si se verifica una de las tres siguientes condiciones:

1) $r = 0$ y $s \in A_o$
2) $0_r << 0_s$
3) $r \in B_*$ y $s = *$

- $D \subset V \times V$ conjunto de arcos disyuntivos, estos arcos relacionan aquellas operaciones que deben ser realizadas en la misma máquina o que comparten uno o más recursos, sin capacidad para atender a las dos simultáneamente.

$(r,s) \in D$ si y solo si se da una de las condiciones:

1) $\#(0_r) = \#(0_s)$
2) 0_r y 0_s comparten un recurso R_h tal que

$$q_{rh} + q_{sh} > q_h$$

Asociamos a cada vértice un vector de pesos, que para cada v_r denotaremos $v_r(i)$ con $i=0,\ldots,s$, $v_r(0)$ contiene el valor de t_r y $v_r(i) = q_{ri}$ cantidad del recurso R_i requerida por la operación O_r .

Convenimos que $v_*(i) = v_o(i) = 0$, $i=0,1,\ldots,s$.

Definimos grafo disyuntivo inverso $G' = (V',C',D')$ con respecto a $G=(V,C,D)$ mediante:

- $V' = V$
- $(r,s) \in C'$ si y solo si se verifica una de las tres condiciones:

  a) $r = 0$ y $(N_n - s + 1, *) \in C$

  b) $(N_n - s + 1, N_n - r + 1) \in C$

  c) $(0, N_n - r + 1) \in C$ y $s = *$

- $(r,s) \in D'$ si y solo si $(N_n - r + 1, N_n - s + 1) \in D$
- $v'_r(i) = v_{N_n - r + 1}(i)$, $i = 0,1,\ldots,s$.

Dado un problema definido por G, podemos construir su problema inverso que vendrá dado por G'. Para algunos criterios de optimalidad estos dos problemas están fuertemente relacionados.

Una de las ventajas de la utilización de la representación basada en el grafo disyuntivo es la de permitir una fácil caracterización de las secuencias factibles.

Un par de arcos $((r,s),(s,r))$ se denomina <<situado>> si uno de los arcos (por ejemplo (r,s)) ha sido añadido a un conjunto $E \subset D$ (inicialmente vacío) de arcos <<elegidos>> y el otro lo ha sido a un conjunto $R \subset D$ de arcos <<rechazados>>. Una vez situados todos los pares de arcos disyuntivos, queda establecida una relación de orden entre las operaciones que deben ser procesadas en la misma máquina o que comparten un mismo recurso insuficiente para satisfacer simultáneamente la demanda de ambas. Para que este orden nos origine una secuencia factible hay que exigir la condición de <<transitividad>> :

Si $(r,s) \in E$ y $(s,t) \in E$ entonces $(r,t) \in E$

Si construimos el disrafo ponderado $G(E) = (V, C \cup E)$ se verificará que la relación de orden dada por E define una secuencia factible si y solo G(E) no contiene ciclos.

Teorema Si $M_1, \ldots, M_m$ pueden ser numeradas de forma que si $O_r < O_{r+1}$ la máquina en que se procesa O_r tiene menor índice que la de O_{r+1}, entonces, dado E, la transitividad en cada máquina es una condición suficiente para que G(E) no contensa ninsún ciclo.

Demostración

Cualquier ciclo contendrá un arco conjuntivo (r,r+1) tal que $\#(O_r) < \#(O_{r+1})$, el tercer vértice del ciclo, s, tendrá que verificar por un lado que $\#(O_{r+1}) \leq \#(O_s)$ y por otro que $\#(O_s) \leq \#(O_r)$ en contradicción con que $\#(O_r) < \#(O_{r+1})$.

[]

Las condiciones del teorema se verifican en el caso de que cada trabajo tensa una sola operación en cada máquina, y el orden en que se procesan los trabajos sea idéntico para todas las máquinas. Problemas en que se cumplen estas condiciones son los denominados <<establecimientos de flujo>> ("flow-shop") .

I.2 Hipótesis del Modelo Básico

La mayoría de los resultados que constituyen la Teoría de la Secuenciación se basan en la suposición de que se verifican ciertas condiciones. A continuación, enumeramos cuáles son las condiciones más frecuentemente impuestas, que dan lugar al denominado <<Modelo Básico>>.

Hipótesis sobre los trabajos

T1) J es conocido y fijo.

T2) Todos los trabajos son independientes y están disponibles en el instante inicial.

T3) Todos los trabajos permanecen disponibles durante un periodo ilimitado.

T4) Cada trabajo puede estar en uno de los tres estados siguientes:

- Esperando ser procesado por la próxima máquina.
- Siendo procesado por una máquina.
- Completado.

T5) Todos los trabajos tienen igual importancia.

T6) Cada trabajo es procesado por todas las máquinas que tiene asignadas.

T7) Cada trabajo es procesado por una sola máquina en cada instante.

Hipótesis sobre las máquinas

M1) M es conocido y fijo.

M2) Todas las máquinas son independientes y están disponibles en el instante inicial.

M3) Todas las máquinas permanecen disponibles durante un periodo de tiempo ilimitado.

M4) Cada máquina puede estar en uno de los tres estados siguientes:

- Esperando el próximo trabajo.
- Procesando un trabajo.
- Parada, por haber completado el último trabajo.

M5) Todas las máquinas tienen igual importancia.

M6) Cada máquina procesa todos los trabajos que tiene asignados.

M7) Cada máquina procesa un solo trabajo en cada instante.

Hipótesis sobre los recursos

R1) R es conocido y fijo.

R2) Todos los recursos están disponibles en el instante inicial y son independientes.

Hipótesis sobre trabajos y máquinas

TM1) Todos los tiempos de procesamiento son fijos e independientes de la secuencia.

TM2) Cada operación una vez comenzada debe ser completada sin interrupción.

TM3) El orden de proceso para cada trabajo es conocido y fijo.

TM4) El orden de proceso para cada máquina es desconocido y debe ser establecido.

Hipótesis sobre trabajos y recursos

TR1) La cantidad de recursos requerido por cada trabajo es fija e independiente de la secuencia.

TR2) Un recurso puede ser compartido por varios trabajos en un instante dado.

A continuación se comentará la influencia que sobre el modelo tiene el que no se satisfaga alguna de las hipótesis, y se introducirá una notación que nos permita indicar qué hipótesis son las que no se asumen para cada problema particular.

Las hipótesis T1 y M1 son fundamentales por cuanto diferencian entre problemas estáticos/determinísticos y dinámicos/estocásticos.

En general, se considera que los problemas determinísticos caen dentro del campo de la Teoría de la Secuenciación y los estocásticos en el de la Teoría de Colas. Sin embargo, esta división no es tajante, y en nuestro estudio el parámetro que determinará el que un problema pertenezca a una u otra área será el número de trabajos " n ". Así, en el caso en que n sea fijo y conocido tendremos un problema de secuenciación.

La hipótesis T2 puede dejar de verificarse por una, o ambas, de las siguientes causas:

a) No todos los trabajos están disponibles en el instante inicial, sino que cada J_i lo está en un tiempo r_i que llamaremos <<tiempo de llegada>>, por uniformidad respecto a la terminología empleada en Teoría de Colas, o bien, <<tiempo de preparación>>. Se utilizará la notación '' $r_i > 0$ '' para indicar estos problemas.

Si existe, a lo más, un r_i estrictamente positivo, esta situación se denotará mediante '' $r_n > 0$ '' .

b) Existe una relación de precedencia " < " en el conjunto de trabajos de forma que $J_i < J_j$ indica que J_j no no puede empezar a ser procesado hasta que lo haya sido J_i completamente.

Esta relación puede ser representada mediante un digrafo acíclico $H = (V , A)$ con vértices $V = \{ 1, \ldots ,n \}$ y arcos $A = \{ (i,j) / J_i < J_j \}$.

Una representación más eficiente viene dada por el <<núcleo transitivo>> $H' = (V , A')$ de H , $A' \subset A$, grafo que verifica:

$$\forall u,v \in V \text{ si } \exists w \in V \text{ tal que } (u,w) , (w,v) \in A' \text{ entonces } (u,v) \in A'$$

Es conveniente distinguir dos casos especiales:

I) Que H' sea una <<ramificación>> , es decir ,un conjunto de árboles dirigidos con grados de incidencia o salida a lo más igual a uno.

Se presentan tres casos:

- grado de incidencia y salida a lo más igual a uno, lo que se denotará `` cadena '' .
- grado de salida a lo más igual a uno: `` arbol → ''.
- grado de incidencia a lo más igual a uno:`` arbol ← ''.

II) Que H' sea un grafo serie-paralelo.

Denotaremos el caso general de existencia de relaciones de precedencia mediante `` prec '' , el caso I mediante `` arbol '' y el II por `` s - p '' .

Frecuentemente T3 debe ser relajada por existir <<fechas de entrega>> d_i (i=1,...,n) para los trabajos. En general, la no completación de un trabajo antes de su fecha de entrega nos producirá un costo. Hay, sin embargo, situaciones en que forzosamente un trabajo J_i debe completarse antes del instante d_i, que denominaremos <<fecha de entrega obligatoria>>, y esta situación la representaremos por `` < $d_i$ '' . Y por `` < d '' cuando todos los trabajos tengan la misma fecha de entrega obligatoria.

A la diferencia $d_i - r_i$ entre la fecha de entrega y el tiempo de llegada del trabajo J_i se le denomina <<tiempo disponible>>.

La mayoría de los resultados en el campo de la Teoría de de la Secuenciación consideran el conjunto de recursos R vacío, por ello indicamos mediante la notación `` rec '' el hecho de ser R no vacío, y por `` rec1 '' la presencia de un único recurso. Existen problemas para los que no se verifica R1, en el sentido de que la disponibilidad de recursos varía con el tiempo, estos problemas se denotarán `` rec var '' y cuando solo existe un recurso y su disponibilidad es variable se representan mediante `` rec1 var ''.

Con respecto a T4 existen situaciones en las que la capacidad de almacenamiento intermedio está limitada, o no existe, por lo que cada trabajo, una vez comenzado, debe ser

procesado sin interrupción hasta que es completado. Esta situación de "no espera" puede deberse a ciertas características técnicas (por ejemplo, los trenes de laminado en los que el metal debe procesarse continuamente a gran temperatura, o los computadores en donde el tamaño del "buffer" sea pequeño y su utilización costosa). Estos problemas se denotarán mediante `` sin espera ''.

Exigiremos que M4 se verifique siempre, debido a que son muy raras las situaciones en que sea rentable tener desocupadas las máquinas mientras quedan trabajos por completar, por lo que no suele convenir asignar las máquinas a tareas alternativas mientras no se haya terminado de procesar el último trabajo.

Si las máquinas tienen distinta importancia relativa, asignaremos un peso conveniente v_K a cada M_K. Y análogamente, se dará un peso w_i a cada J_i cuando no se verifique la hipótesis T5.

Las hipótesis T6 y M6 no pueden ser incumplidas si queremos mantenernos dentro del campo cubierto por la Teoría de la Secuenciación.

T7 y M7 ,sin embargo, excluyen problemas que pueden ser tratados sin demasiada dificultad, tales como los procesos de ensamblaje. Estos pueden incluirse dentro del esquema de representación dado por el grafo disyuntivo, sin más que sustituir una cadena de operaciones ,que constituyen un trabajo, por un árbol dirigido con grado de salida , a lo más, uno. No admitiremos más relajaciones en las hipótesis T7 y M7, por cuanto que aquellas situaciones en las que en la literatura se admite que una máquina pueda procesar más de un trabajo a la vez, caen dentro de nuestro modelo sin más que observar que tales máquinas no son tales, sino que deben de englobarse dentro del conjunto R de recursos. Dentro de este tipo de problemas entra la importante clase de los de <<Secuenciación de Proyectos con Recursos Limitados>>.

Una situación particular es aquella en que exista uno o más recursos R_h con capacidad para atender a los n trabajos simultáneamente, es decir,

$$q_h > \sum_{i=1}^{n} q_{ih}$$

este caso se denotará '' $q_h >$ demanda '' .

En numerosas situaciones reales los tiempos de preparación de un trabajo dependen del tipo del que haya sido completado inmediatamente antes. Por tanto, el tiempo total de procesamiento dependerá del lugar que ocupe el trabajo en la secuencia, este tipo de problemas se representará '' dep sec ''.

Existen situaciones particulares con respecto a los tiempos de procesamiento que por su incidencia en las técnicas de resolución son convenientes señalar mediante una notación particular, así si los tiempos de procesamiento son unitarios pondremos '' $t_{ij} = 1$ '' y cuando están acotados entre dos valores, cotas inferior y superior, denotaremos '' $\underline{t} < t_{ij} < \overline{t}$ '' .

Por otra parte puede ocurrir que los tiempos de procesamiento de una operación dependan de la máquina en que este proceso se lleve a cabo. En estas circunstancias, serán necesarios tres subíndices para denotar los tiempos. Así, t_{iJK} será el tiempo que se necesita para procesar la operación J del trabajo i en la máquina K, ($i=1,\ldots,n$; $J=1,\ldots,n_i$; $K=1,\ldots,m$).

En el caso más general, no existirá relación entre los t_{iJK} para los distintos valores de K. Sin embargo, podemos citar un caso en que esto no es así ; decimos que las máquinas son <<uniformes>> si se verifica que $t_{iJK} = P_K t_{ij}$, donde P_K es un factor que depende de M_K, y que puede ser

considerado un índice de velocidad. Es obvio que, en este caso

$$t_{iJK} = \frac{P_K}{P_{K'}} t_{iJK'} .$$

Los problemas en los que los tiempos dependan de las máquinas se representarán por '' maq dep '' . Y si éstas son uniformes se pondrá '' maq unif ''.

Existen casos en los que los trabajos pueden ser acortados a expensas de un coste adicional, en estas situaciones los tiempos no son fijos pero sí controlables. Se denotarán mediante '' t var ''.

Una más drástica relajación de la hipótesis TM1 es la de admitir que los tiempos de procesamiento son aleatorios. Los problemas en que ésto ocurre caen fuera del área de la Teoría de la Secuenciación Determinística y son muy escasos los resultados que se conocen. En el caso estocástico hay que distinguir dos tipos básicos de políticas:

- Políticas estáticas. Son aquellas en las que la decisión sobre la regla a seguir para ejecutar los n trabajos se toma en el instante $t = 0$ y no cambia a lo largo del proceso.
- Políticas dinámicas. La regla de ordenación varía con el tiempo a medida que se dispone de nueva información.

En lo que sigue, denotaremos el carácter aleatorio de cualquier parámetro del modelo mediante un asterisco "*". Así, la aleatoriedad de los tiempos de procesamiento la denotaremos por t_i^* .

Cuando admitamos la posibilidad de interrupción de una operación, antes de su completación, lo indicaremos mediante '' interrup '' . En principio, suponemos que cuando una operación es interrumpida, al ser reanudada, se sigue sin tener que repetir la parte ya procesada. Hay, sin embargo, situaciones en que interrumpir una operación nos obliga a repetir desde el comienzo el procesamiento de la misma. Si los tiem-

pos de procesamiento son aleatorios, puede ocurrir que el tiempo de procesamiento en la repetición sea igual al que habría sido si no hubiera existido la interrupción, o bien distinto. En cualquier caso, esta situación se denotará por `` int rep '' .

Destacamos también una situación que no supone el incumplimiento de ninguna hipótesis, pero que al igual que los casos $t_{ij} = 1$ y $\underline{t} < t_{ij} < \bar{t}$ tienen particular influencia sobre la estructura del problema. Este caso es aquel en que el número de operaciones para todos los trabajos está limitado por una cota superior. Recurriremos en este caso a la notación `` $n_j < \bar{n}$ '' .

Por último, las hipótesis TM3 y TM4 son fundamentales. Y por la diferencia establecida entre recursos y máquinas, TR2 tampoco puede incumplirse.

Teorema S es un conjunto dominante

Demostración

Sea s una secuencia optimal que no pertenezca a S. Si es posible disminuir el tiempo de comienzo de alguna operación, manteniendo la factibilidad y el mismo orden de procesamiento en cada máquina, ésto no incrementará el valor de $S_r + t_r$ para ningún r, en particular para las operaciones finales de cada trabajo, para las cuales $C_i = S_r + t_r$. Por tanto no aumentará ningún tiempo de completación y, por lo mismo, el valor de cualquier función regular.
Cuando ya no sea posible disminuir ningún S_r, tendremos un elemento de S, cuyo costo no será mayor que el de s. Esto demuestra el teorema.

[]

La mayor parte de los criterios suponen la existencia de unas funciones de costo $c_i(t)$ (i=1,...,m), no decrecientes en la varible tiempo t, y pueden ser clasificados en dos grandes tipos:

- Minimización del máximo costo

$$c_{max} = \max_i \{ c_i(C_i) \}$$

- Minimización del costo total

$$\sum_{} c_i = \sum_{i=1}^{n} c_i(C_i)$$

Existen algunos resultados para funciones no decrecientes arbitrarias $c_i(t)$, y en el apartado III.1 se estudia el caso de ser el criterio función del nivel de recursos consumidos. Sin embargo, los criterios más frecuentemente estudiados son los basados en funciones $c_i(t)$ lineales, o lineales a trozos. Entre las de estos tipos podemos señalar:

- Tiempo de completación (C_i)
- Flujotiempo (F_i), tiempo en que está el trabajo en el sistema cuando no se verifica la hipótesis T2.

$$F_i = C_i - r_i$$

Y los del segundo tipo son los de minimización de la:

- Suma de los tiempos de completación ($\sum_{i} C_i$)
- Suma de los flujotiempos ($\sum_{i} F_i$)
- Suma de los desfases ($\sum_{i} L_i$)
- Suma de las tardanzas ($\sum_{i} T_i$)
- Suma de los adelantos ($\sum_{i} E_i$)
- Suma de los tiempos de espera ($\sum_{i} W_i$)
- Número de trabajos tardíos ($\sum_{i} U_i$)

Todos los criterios anteriores son regulares excepto los basados en el adelanto.

Si la hipótesis T6 es relajada, se obtienen las versiones ponderadas de los criterios del segundo tipo, así, por ejemplo, tendremos:

- Suma ponderada de los tiempos de completación ($\sum_{i} w_i C_i$)

En caso de ser $w_i = 1/n$, $i=1,\ldots,n$, el criterio será la minimización del tiempo medio de completación :

$$\bar{C} = \frac{1}{n} \sum_{i} C_i$$

Además de los criterios anteriores,basados en los tiempos de completación y las fechas de entrega, es interesante trabajar con algunos basados en el nivel de utilización del sistema y en los costes de inventario, entre ellos:

- Número de trabajos que están siendo procesados en el instante t , $N_P(t) = | J_P(t) |$

  donde $J_P(t) = \{ J_i \mid \exists O_r \in J_i : S_r \leq t \leq S_r + t_r \}$

- Número de trabajos que están esperando en el instante t,

$$N_e(t) = |J_e(t)|$$

siendo $J_e(t) = \{ J_i \mid \exists\, O_r, O_{r+1} \in J : S_r + t_r < t < S_{r+1} \}$

- Número de trabajos finalizados en el tiempo t,

$$N_f(t) = |J_f(t)|$$

siendo $J_f(t) = \{ J_i \mid C_i < t \}$

- Cantidad de trabajo en marcha en el tiempo t,

$$A_P(t) = \sum_{S_r \le t \le S_r + t_r} t_r$$

Se verifica que en todo instante t,

$$N_P(t) + N_e(t) + N_f(t) = n$$

Y se denomina <<número de trabajos en el sistema>> a la cantidad $N_s(t) = N_P(t) + N_e(t) = n - N_f(t)$

Cuando queremos disminuir los costos de inventario de los trabajos en marcha , el criterio será minimizar

$$\bar{N}_e = \frac{1}{C_{max}} \int_0^{C_{max}} N_e(t)\, dt$$

o bien

$$\bar{N}_s = \frac{1}{C_{max}} \int_0^{C_{max}} N_s(t)\, dt$$

Si se pretende disminuir los costos de inventario de los trabajos completados, se tratará de minimizar

$$\bar{N}_f = \frac{1}{C_{max}} \int_0^{C_{max}} N_f(t)\, dt$$

En las situaciones en que se desea un uso eficiente de las máquinas disponibles, el objetivo será maximizar

$$\bar{N}_P = \frac{1}{C_{max}} \int_0^{C_{max}} N_P(t)\, dt$$

o bien

$$\bar{A}_P = \frac{1}{C_{max}} \int_0^{C_{max}} A_P(t)\, dt$$

Otra forma de buscar una eficiente utilización de las máquinas, es el empleo de los criterios basados en el <<tiempo de desocupación>> I_K de M_K, que se define:

$$I_K = C_{max} - \sum_{O_r \in M_K} t_r \qquad (K=1,\ldots,m)$$

Los criterios son:

- Minimizar el tiempo total de desocupación, $\sum I_K$
- Minimizar la suma ponderada de los tiempos de desocupación

$$\sum v_K I_K$$

- Maximizar la utilización media

$$\bar{U} = \frac{\sum_{i=1}^{n} \sum_{O_r \in J_i} t_r}{m\, C_{max}}$$

Relaciones entre criterios

Decimos que dos criterios son <<equivalentes>> si cualquier secuencia que es óptima con respecto a uno de ellos también lo es respecto del otro.

A continuación daremos una serie de resultados que establecen la equivalencia entre grupos de criterios.

Teorema Los siguientes criterios son equivalentes:

1) Minimizar C_{max}
2) Maximizar $\bar{N}_P$
3) Maximizar $\bar{A}_P$
4) Minimizar $\sum I_K$

5) Minimizar $\sum_{k} v_k I_k$

6) Maximizar $\bar{U}$

Demostración

1) <===> 2)

$$\bar{N}_P = \frac{1}{C_{max}} \int_0^{C_{max}} N_P(t)\,dt = \frac{1}{C_{max}} \sum_{i=1}^{n} \sum_{O_r \in J_i} t_r$$

(constante)

1) <===> 3)

$$\bar{A}_P = \frac{1}{C_{max}} \int_0^{C_{max}} A_P(t)\,dt = \frac{1}{C_{max}} \sum_{i=1}^{n} \sum_{O_r \in J_i} t_r^2$$

(constante)

1) <===> 4) <===> 5)

$$\sum_{k=1}^{m} I_k = \sum_{k=1}^{m} \left(C_{max} - \sum_{O_r \in M_k} t_r \right) = m\,C_{max} - \sum_{k=1}^{m} \sum_{O_r \in M_k} t_r$$

(constantes)

$$\sum_{k=1}^{m} v_k I_k = \left(\sum_{k=1}^{m} v_k \right) C_{max} - \sum_{k=1}^{m} v_k \sum_{O_r \in M_k} t_r$$

(constantes)

1) <===> 6)

Es inmediata a partir de la definición. []

Teorema Los siguientes criterios son equivalentes:

1) $\sum_{i} C_i$ 2) $\sum_{i} F_i$ 3) $\sum_{i} W_i$ 4) $\sum_{i} L_i$

5) $\bar{C}$ 6) $\bar{F}$ 7) $\bar{W}$ 8) $\bar{L}$

(En todos ellos el objetivo es minimizar)

Demostración

1) <===> 2) <===> 3) <===> 4)

$$\sum_{i=1}^{n} C_i = \sum_{i=1}^{n} F_i + \sum_{i=1}^{n} r_i = \sum_{i=1}^{n} W_i + \sum_{i=1}^{n} r_i + \sum_{i=1}^{n} \sum_{O_r \in J_i} t_r =$$

$$= \sum_{i=1}^{n} L_i + \sum_{i=1}^{n} d_i$$

Las sumas son equivalentes a los valores medios por diferir en una constante. []

Análogamente podemos establecer el siguiente resultado

Teorema Los siguientes criterios son equivalentes:

1) $\sum w_i C_i$ 2) $\sum w_i F_i$ 3) $\sum w_i W_i$ 4) $\sum w_i L_i$

[]

Teorema Si una secuencia es óptima respecto a L_{max} entonces es óptima respecto de T_{max}

Demostración

Si $L_{max} = \max_i \{ L_i \} \leq L'_{max} = \max_i \{ L'_i \}$

entonces

$$T_{max} = \max_i \{ \max(0, L_i) \} = \max \{ 0 , L_{max} \}$$

$$\leq \max \{ 0 , L'_{max} \} = T'_{max}$$

[]

Análogamente puede probarse que si una secuencia minimiza L_{max} entonces maximiza E_{max}.

Teorema Son equivalentes: 1) $\bar{N}_s$ 2) $\bar{N}_f$ 3) $\bar{C} / C_{max}$

Demostración

1) <===> 2) $N_s = n - N_f$

1) <===> 3)

$$\bar{N}_s = \frac{\sum C_i}{C_{max}} = n \bar{C} / C_{max}$$

[]

Teorema Son equivalentes: 1) $\bar{N}_e$ 2) $\bar{W} / C_{max}$

Demostración

$$\bar{N}_e = n \bar{C} / C_{max} - \bar{N}_p = n \bar{C} / C_{max} - \left(\sum_{i=1}^{n} \sum_{r \in J_i} t_r \right) / C_{max} =$$

$$= n \bar{W} / C_{max}$$

[]

Si $m > 1$, $\bar{N}_s$, $\bar{N}_f$, $\bar{C} / C_{max}$, $\bar{N}_e$ y $\bar{W} / C_{max}$ no son medidas regulares.

Para el caso de una sola máquina, ($m = 1$) , $\bar{N}_s$, $\bar{N}_f$ y $\bar{N}_e$ son equivalentes a $\bar{C}$.

Como compendio de los resultados anteriores, tenemos que los criterios que hemos definido pueden resumirse en los de minimización de las siguientes medidas regulares:

- Calendario
- Suma ponderada de los tiempos de completación
- Desfase máximo
- Suma ponderada de las tardanzas
- Número ponderado de trabajos tardíos

Criterios múltiples

Hay numerosas situaciones en las que no existe un único objetivo, supongamos, por ejemplo, que tenemos dos criterios $f_1(s)$ y $f_2(s)$ y queremos optimizar ambos. Esta situación puede abordarse desde dos puntos de vista:

- Optimización Lexicográfica, ésto supone destacar una medida fundamental f_1 , para valorar las secuencias, y una medida secundaria f_2 , que nos permite seleccionar una secuencia entre aquellas que optimizan la medida fundamental.

Este tipo de criterios los denotaremos $f_2 \mid f_1$.

- Optimización Conjunta, al no existir, en general, una secuencia s que optimize ambos criterios simultáneamente nos restringimos a las secuencias <<eficientes>> , que son aquellas que verifican que no existe ninguna otra secuencia s' tal que $f_1(s') \leq f_1(s)$

$f_2(s') \leq f_2(s)$ con alguna de las desigualdades estricta.

(suponemos que estamos minimizando).

Estos problemas se denotarán f_1 , f_2.

Lógicamente, tanto en el caso de optimización lexicográfica como en el de la conjunta, no hay razón para restringirnos a dos criterios, en caso de existir N de ellos la notación que emplearemos será $f_1 \mid f_2 \mid \ldots \mid f_N$ o $f_1 , f_2 , \ldots , f_N$ respectivamente.

Por otro lado, existen problemas en los que se exige que un subconjunto E de trabajos verifiquen unas determinadas condiciones, tales como, por ejemplo, estar completados antes de su fecha de entresa. Si el criterio a optimizar es el f , estos problemas se denotarán $f \mid E$.

Todo lo anterior es válido siempre que los parámetros del modelo sean determinísticos. En el caso estocástico, cambia radicalmente el concepto de optimalidad, puesto que los valores que toma la medida a optimizar seguirán una distribución de probabilidad que dependerá de las distribuciones de los parámetros del modelo. En este sentido, cada posible combinación de valores de los parámetros determinará un problema determinístico. En general, los problemas estocásticos se abordan eligiendo un problema determinístico que represente al conjunto de todos los generados por el estocástico, sin tener que ser necesariamente uno de ellos.

Frecuentemente se toma como problema representante el obtenido al tomar valores esperados. Por ejemplo, minimizar la esperanza del flujotiempo ponderado, $E[\sum w_i F_i]$. Otro enfoque consiste en tomar como representante la minimización de la probabilidad de que la medida del problema estocástico exceda de un determinado valor. Por ejemplo, $P[T_{max} > 0]$ probabilidad de que exista algún trabajo tardío.

I.4 Clasificación de los problemas

En el presente apartado desarrollamos un esquema de clasificación que extiende el inicial de Conway,Maxwell, Miller (1967) y los posteriores, basados en el anterior,de Rinnooy Kan (1976), Graham,Lawler,Lenstra,Rinnooy Kan (1978) y French (1982).

La representación de un problema genérico vendrá dada por $a \mid b \mid c , I \mid d$
donde cada uno de los cinco parámetros tiene el siguiente significado:

a - Número de trabajos

b - Número de máquinas. Se distinguirá entre los casos en que b sea igual a uno, dos o tres y aquellos otros en que sea una variable entera m.

c - Tipo de procesamiento de los trabajos en las máquinas, distinguiremos los siguientes casos

P = máquinas en paralelo

F = establecimiento de flujo ("flow-shop"), cada trabajo está constituido por m operaciones, $J_i = \{ O_{i1} , \ldots , O_{im} \}$ $i=1,\ldots,n$,tales que

$$O_{i1} < O_{i2} < \ldots < O_{im}$$

y teniendo que ser procesada O_{iK} en la maquina M_K.

O = establecimiento abierto ("open-shop") , lo mismo que en el caso anterior, cada trabajo está formado por m operaciones, teniendo que procesarse la operación K en la máquina M_K. Sin embarso, en este caso no importa el orden de procesamiento de las operaciones.

J = establecimiento de trabajos ("Job-shop"), cada trabajo J_i está constituido por n_i operaciones,

$$J_i = \{ O_{i1}, \ldots, O_{in_i} \},$$

cada operación O_{iK} se procesa en la máquina $\#(O_{iK})$ durante un tiempo t_{iK}, debiéndose verificar que $\#(O_{i\,K-1})$ sea distinta de $\#(O_{iK})$, (es decir, no se pueden procesar dos operaciones consecutivas de un mismo trabajo en la misma máquina).

En el caso de ser b=1, el parámetro c no se indica.

I - Es un conjunto de parámetros (posiblemente vacío) que indica las hipótesis incumplidas, tal como se desarrolló en el apartado I.2.
Los elementos posibles de I son:

$r_i > 0$	Los tiempos de preparación son distintos.
$r_n > 0$	A lo más existe un $r_i > 0$.
prec	Existe una relación de precedencia entre los trabajos.
arbol	Grado de incidencia o salida a lo más igual a uno.
cadena	Grado de incidencia y salida a lo más igual a uno.
arbol→	Grado de salida a lo más igual a uno.
arbol←	Grado de incidencia a lo más igual a uno.
s-p	La relación de precedencia es del tipo serie-paralelo.
$\leq d_i$	Las fechas de entresa deben ser respetadas forzosamente.

J = establecimiento de trabajos ("job-shop"), cada trabajo J_i está constituido por n_i operaciones,

$$J_i = \{ O_{i1} , \ldots , O_{in_i} \} ,$$

cada operación O_{ik} se procesa en la máquina $\#(O_{ik})$ durante un tiempo t_{ik}, debiéndose verificar que $\#(O_{ik})$ sea distinta de $\#(O_{i\,k-1})$, (es decir, no se pueden procesar dos operaciones consecutivas de un mismo trabajo en la misma máquina).

En el caso de ser b=1 , el parámetro c no se indica.

I - Es un conjunto de parámetros (posiblemente vacío) que indica las hipótesis incumplidas, tal como se desarrolló en el apartado I.2 .
Los elementos posibles de I son:

$r_i > 0$	Los tiempos de preparación son distintos.
$r_n > 0$	A lo más existe un $r_i > 0$.
prec	Existe una relación de precedencia entre los trabajos.
arbol	Grado de incidencia o salida a lo más igual a uno.
cadena	Grado de incidencia y salida a lo más igual a uno.
arbol→	Grado de salida a lo más igual a uno.
arbol←	Grado de incidencia a lo mas igual a uno.
s-p	La relación de precedencia es del tipo serie-paralelo.
$\leq d_i$	Las fechas de entresa deben ser respetadas forzosamente.

$\leq d$	La fecha de entrega obligatoria es la misma para todos los trabajos.
rec	El conjunto R de recursos es no vacío.
rec var	La disponibilidad de los recursos varía con el tiempo.
rec1 var	Existe un único recurso y su disponibilidad es variable.
$q_h > demanda$	El recurso R_h es capaz de atender a los n trabajos simultáneamente.
sin espera	Los trabajos, una vez comenzados, deben ser procesados sin interrupción hasta su completación.
dep sec	Los tiempos de comienzo de los trabajos dependen de la secuencia.
$t_{ij} = 1$	Tiempos de procesamiento unitarios.
$\underline{t} < t_{ij} < \bar{t}$	Tiempos de procesamiento acotados.
maq dep	Tiempos de procesamiento dependientes de las máquinas.
t var	Tiempos de procesamiento variables.
maq unif	Las máquinas son uniformes.
interrup	Se admite la interrupción de una operación aunque no haya sido completada.
int rep	Las operaciones interrumpidas deben ser procesadas de nuevo desde su inicio.
$n_j < \bar{n}$	Número de operaciones por trabajo acotada.
*	Un asterisco sobre cualquier parámetro del modelo, denota que éste es aleatorio.

d - Indica el criterio de optimalidad, según la notación del apartado I.3 .

Ejemplos

$n \mid 1 \mid prec \mid \sum w_i U_i$

Representa un problema con n trabajos, una máquina, existe una relación de precedencia entre los trabajos y el criterio de optimalidad es el número ponderado de trabajos tardíos.

$n \mid 3 \mid P, t_{ij} = 1 \mid C_{max}$

Minimización del calendario para un problema con tres máquinas en paralelo , n trabajos y tiempo de procesamiento unitarios.

En general, por "complejidad" de un algoritmo que resuelve un determinado problema, entendemos la correspondencia que nos da el tiempo, o la cantidad de memoria, requerido por el mismo para la obtención del resultado final, en función del tamaño n de las configuraciones particulares del problema.

El tamaño de una configuración, cuando estamos utilizando un ordenador para la resolución del problema, dependerá del esquema de representación, o codificación, que nos permite expresar la configuración como una secuencia de símbolos sobre algún alfabeto fijo tal como bits, caracteres ASCII o EBCDIC. Con lo cual podemos definir tamaño de la entrada como el número de símbolos de su representación. El tamaño dependerá de parámetros del problema tales como el número de trabajos o máquinas.

Para representar la función que nos da la complejidad de un problema, se introduce la notación de orden de magnitud O(.) . Dadas $f, g : Z^{+} \longrightarrow R^{+}$, decimos que una función f(n) es O(g(n)) si existe una constante $c \neq 0$ tal que

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} \leq c$$

y se dice que f(n) es o(g(n)) si

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} = 0$$

Un problema podrá considerarse "fácil" si podemos encontrar un "buen" algoritmo que lo resuelva. Es Edmonds (1965) el primero en definir como buenos algoritmos aquellos cuyo tiempo de ejecución es polinomial, esto es o(P(n)) para algún polinomio P(n). Se suele dar el nombre de <<eficientes>> a tales algoritmos que pueden realizarse en tiempo polinomial. Como ejemplos de problemas para los que se conocen algoritmos eficientes podemos citar $n|1|\sum w_j C_j$ y $n|1|L_{max}$, que por consistir esencialmente en la ordenación de n números utilizando solamente comparaciones binarias tienen una complejidad $O(n \log_2 n)$.

Existen, sin embargo, numerosos problemas "difíciles" para los que no se conocen algoritmos eficientes, y para los que se conjetura la no existencia de tales algoritmos. Hay que citar como ejemplo el problema general de programación entera. Para estos problemas, los algoritmos, en general, tienen una estructura enumerativa, lo que implica una búsqueda a lo largo de un árbol cuya profundidad (número de niveles) es O(p(n)), por lo cual su complejidad en el peor caso será exponencial en n.

Si consideramos informalmente la clase P como la formada por todos los problemas resolubles en tiempo polinomial y la clase NP de problemas resolubles mediante una búsqueda sobre un árbol de profundidad polinomial, es obvio que P c NP . La cuestión de si P = NP , es uno de los problemas abiertos más importantes de las matemáticas actuales, desde que empezó a ser estudiada por Cook (1971) y Karp (1972).

Cook demuestra que todo problema en NP puede "reducirse" al problema de la satisfactibilidad, que consiste en encontrar si una expresión en variables booleanas puede tomar el valor "verdadero" . Por lo que la cuestión de si P = NP es equivalente a la existencia de un buen algoritmo para el problema de la satisfactibilidad. El mismo razonamiento es aplicable a los problemas en NP denominados NP-completos [Knuth (1974)] que son equivalentes en el siguiente sentido:

- Para ninguno de ellos se conocen algoritmos eficientes
- Si alguno perteneciera a P , entonces todos los problemas en NP serían resolubles mediante algoritmos eficientes y por tanto P = NP.

La NP-completitud de un problema es generalmente aceptada como evidencia de la no existencia de algoritmos eficientes y como justificación, por tanto, del empleo de métodos enumerativos tales como los de ramificación y acotación o los de programación dinámica.

II.1 Elementos de la Teoría de la Complejidad

El primer elemento que debemos establecer para un desarrollo formal de la Teoría de la Complejidad es el sistema de cómputo teórico sobre el que trabajaremos, puesto que la cantidad de tiempo o almacenamiento requerido por un algoritmo dependerá del modelo de computador elegido.

El modelo abstracto de computador más antiguo y estudiado es el de la máquina de Turing, modelo introducido por Turing en 1936. Sobre este modelo básico y sus variantes se han desarrollado gran parte de los trabajos existentes en este campo. La variante más comúnmente utilizada es la máquina de Turing determinística con una sola cinta y a ella nos referiremos simplemente con el nombre de <<máquina de Turing>>. Está constituida por un <<control>> , también denominado <<control finito de estados>> , una <<cinta>> y una <<cabeza de lectura-escritura>> que enlaza ambos elementos.

La cinta está dividida en cuadrados,es infinitamente larga en ambos sentidos, y cada cuadrado es capaz de contener un <<símbolo>> perteneciente a un alfabeto A que contiene un subconjunto E constituido por los símbolos de entrada y un símbolo distinguido B denominado <<blanco>>.

El control en cada instante está en un único estado q perteneciente al conjunto finito de estados Q. Elementos especiales de Q son:

- q_0 estado inicial
- q_S y q_N estados de parada.

La cabeza puede registrar en cada momento un único cuadrado de la cinta.

Antes de describir el funcionamiento de una máquina de Turing es preciso definir lo que entendemos por <<problema>> y <<esquema de codificación>>.

Un problema (de decisión) es una pregunta que tiene una respuesta "sí" o "no" . Normalmente la pregunta tiene varios <<parámetros>> ,es decir, variables libres. Una selección de valores para los parámetros determina una <<configuración>> del problema.

Un esquema de codificación e es una correspondencia entre configuraciones y cadenas de símbolos de E. Al conjunto formado por las cadenas finitas de elementos de E lo denotaremos E^*.

El esquema de codificación en principio puede ser cualquiera, aunque lógicamente se deben exigir ciertas condiciones de "concisión",en el sentido de no incurrir en redundancias, y de "decodificabilidad", de forma que dada la codificación de una configuración seamos capaces de obtener en tiempo polinomial una descripción de cada componente de la configuración.

En función del criterio seguido para la codificación de los números podemos hacer una división en dos grupos; el esquema se denomina <<binario>> si cualquier entero n positivo se representa mediante una cadena de ceros y unos, y <<unitario>> si se representa mediante una cadena de n unos.

El <<tamaño>> de una configuración es la longitud de la cadena que le asigna el esquema de codificación (para una cadena x, su longitud se denotará |x|). Suponemos que el alfabeto A es fijo para el problema e independiente de las configuraciones.

La <<entrada>> para la máquina de Turing será una cadena $x \in E^*$ que se situará en la cinta desde el cuadrado 1 al |x| . Los restantes cuadrados contendrán al principio el símbolo blanco.

Un <<programa>> está determinado por una función de transición

$$P : (Q - \{q_S , q_N\}) \times A \longrightarrow Q \times A \times \{-1,+1\} ,$$

el programa comienza su operación en el estado q_o, con la cabeza de lectura-escritura resistrando el cuadrado 1 y actúa paso a paso de la siguiente forma:

- si el estado en un paso dado es q_S o q_N el programa se detiene y la respuesta es "sí" o "no" respectivamente
- si $q \in Q - \{q_S , q_N\}$ y el símbolo resistrado en el cuadrado examinado es s , se define P(q,s) cuyo valor será (q',s',^) . La cabeza de lectura-escritura borra s, escribe s' en su lusar y se mueve un lusar a su izquierda si ^ = −1 , o la derecha si ^ = +1. Y así sucesivamente.

Dada la cadena x decimos que el programa P <<acepta>> x si y solo si se detiene en el estado q_s.

El <<lenguaje>> L reconocido por P está definido por

$$L_P = \{ x \in E^* \mid P \text{ acepta } x \} .$$

Definiremos <<algoritmo>> como aquel programa que se detiene para cualquier cadena posible sobre el alfabeto de entrada E.

Un programa P <<resuelve>> un problema $\overline{\Pi}$ bajo un esquema de codificación e si es un algoritmo.

El <<tiempo>> utilizado en la computación de un programa para una entrada x es el número de etapas desde el comienzo hasta alcanzar uno de los estados de detención. La <<función de complejidad del tiempo>> está dada por :

$$T_P : Z^+ \longrightarrow Z^+$$

donde $T_P(n)$ es el máximo tiempo de computación para una entrada de lonsitud n.

Un programa P se denomina <<polinomial>> si existe un polinomio p tal que $T_P(n) = O(p(n))$.

Y se define la clase P de la siguiente forma:

$$P = \{ L \mid \exists\, P \text{ polinomial para el cual } L = L_P \}$$

Intuitivamente, la clase P contiene a los problemas resolubles en tiempo polinomial, la formalización de este concepto la hemos basado en el de máquina de Turing determinística, el cuál constituye un modelo razonable de ordenador. Para introducir el concepto de clase NP debemos recurrir al de ordenador no determinístico, esta herramienta de cálculo abstracta podemos imaginarla como un ordenador normal que tuviera una instrucción tal que al ejecutarse produjera instantáneamente dos copias del computador con el mismo estado que tuviese en ese momento. Cada una de las copias cambia entonces de estado independientemente, después de la ejecución de n instrucciones de este tipo, existirán 2^n copias del ordenador, pudiendo estar todas en diferentes estados. Un computador no determinístico aceptaría una entrada si al menos una de las copias se detiene en el estado q_S, y el tiempo será el mayor número de etapas entre las realizadas por todas las copias. Pasamos a continuación a formalizar los conceptos anteriores.

Una <<máquina de Turing no determinística>> es una máquina de Turing a la que se añade un <<módulo conjeturador>> ("guessing") conectado con la cinta mediante una <<cabeza de solo escritura>> (la anterior no es la definición estándar,sino una modificación de la misma debida a Garey & Johnson 1979).

Un programa no determinístico difiere de uno determinístico en dos etapas:

- La etapa inicial es la de <<conjetura>>. Al principio la cadena de entrada x se escribe en los cuadrados 1 al |x| de la cinta y los restantes se rellenan con blancos. La cabeza de lectura-escritura está registrando el primer cuadrado, la de solo escritura el -1 y el control está inactivo.

El módulo conjeturador dirige la cabeza de solo escritura y en cada paso puede realizar una de las siguientes acciones:

- Escribe algún símbolo $s \in A$ sobre el cuadrado que está registrando.
- Se mueve un cuadrado a la izquierda.
- Se detiene, quedando en este caso el módulo inactivo y el control activado en el estado q_o.

La elección de s es totalmente aleatoria. Por tanto el módulo puede construir cualquier cadena de E^* antes de detenerse, o bien no detenerse nunca.

- La etapa de <<chequeo>> comienza cuando el control es activado en el estado q_o, desde este instante el programa actúa como en el caso determinístico. Y se detendrá si llega a uno de los estados q_S o q_N, en caso de que sea el q_S se dirá que el cómputo ha sido de <<aceptación>>.

Existirán, por tanto, un número infinito de posibles cómputos para una cadena de entrada x dada. Decimos que el programa no determinístico acepta x si al menos uno de los cómputos es de aceptación. Análogamente al caso determinísco, el lenguaje reconocido por un programa será el conjunto de cadenas sobre el alfabeto de entrada aceptadas por él. Y el tiempo requerido por el programa no determinístico para aceptar una entrada se define como el mínimo número de pasos en la etapa inicial y de chequeo,entre los cómputos del programa que aceptan la entrada. La función de complejidad del tiempo es en este caso:

$$T_{\mathbb{P}}(n) = \max \{ 1, \{m \mid \exists x \in L_{\mathbb{P}} \text{ con } |x|=n \text{ con tiempo de aceptación por } \mathbb{P} \text{ igual a } m\} \}.$$

Con los elementos anteriores estamos en condiciones de definir la clase NP,

$$NP = \{ L \mid \exists\, \mathbb{P} \text{ no determinístico para el cual } L_{\mathbb{P}} = L \}$$

Después de definir las clases P y NP , llegamos a la noción de NP-completitud; este concepto está ligado estrechamente al de reducibilidad polinomial. Se dice que un problema Π_1 se reduce polinomialmente a un problema Π_2 (y se escribe $\Pi_1 \propto \Pi_2$) si el problema Π_1 puede ser resuelto por un algoritmo A que verifica las siguientes propiedades:

- Llama a una subrutina A' ,que resuelve el problema Π_2.
- Aparte del tiempo consumido en las llamadas a A' , el algoritmo toma un tiempo que es polinomial en el tamaño de la entrada de A.

Si $\Pi_1 \propto \Pi_2$ y Π_2 es resoluble mediante un algoritmo de tiempo polinomial, también existe un algoritmo polinomial para Π_1 . Este resultado puede razonarse fácilmente, si $p_1(n)$ es el tiempo consumido por A' y $p_2(n)$ el que emplea A aparte de las llamadas a A', no podrá haber más de $p_2(n)$ llamadas a A' , y ningún argumento de A' tendrá una longitud mayor que $p_2(n)$,puesto que un tal argumento requeriría más tiempo para ser manejado. Por tanto, cada llamada a A' no empleará un tiempo superior a $p_1(p_2(n))$. Además como no existen más de $p_2(n)$ de estas llamadas, se deduce que el tiempo total consumido por A no excederá de $p_2(n) + p_2(n)\, p_1(p_2(n))$.

La reducibilidad de un problema Π' en otro Π puede verse como si dada una confisuración de Π' es construible una confisuración de Π , de forma que resolviendo Π también se resuelve Π'.

Para introducir risurosamente el concepto de reducibilidad polinomial hay que partir del nivel de lenguajes, así se define <<transformación polinomial>> de un lenguaje $L_1 \subset E_1^*$ en otro $L_2 \subset E_2^*$ como una función

$$f: E_1^* \longrightarrow E_2^*$$

que satisfaga:

- Existe un programa de tiempo polinomial que computa f [*].

- $\forall x \in E_1^*$, $x \in L_1$ si y solo si $f(x) \in L_2$

Si tal transformación existe, escribiremos $L_1 \propto L_2$ y se dirá que L_1 se transforma en L_2.

De la anterior definición se deduce que:

- Si $L_1 \propto L_2$ y $L_2 \in P$ ===> $L_1 \in P$
- Si $L_1 \propto L_2$ y $L_2 \propto L_3$ ===> $L_1 \propto L_3$

Con ésto ya estamos en condiciones de definir el elemento fundamental dentro de la Teoría de la Complejidad. Un lenguaje L se denomina <<NP-completo>> si $L \in NP$ y $\forall L' \in NP$ se verifica que $L' \propto L$.

De la transitividad de la transformación polinomial se deduce que dados L_1, $L_2 \in NP$ entonces:

- si L_1 es NP-completo y $L_1 \propto L_2$ ===> L_2 es NP-completo

Las definiciones formales correspondientes a los problemas de decisión son inmediatas sin más que tener en cuenta que deben referirse a sistemas de codificación determinados, y siguen siendo válidos para ellos los resultados previos.

Si algún problema NP-completo fuera resoluble mediante un algoritmo polinomial, también lo serían todos los demás elementos de NP, y por tanto P = NP, pero a pesar de los grandes esfuerzos realizados durante la última década, no ha sido posible encontrar ningún algoritmo eficiente para problemas NP-completos.

[*] La computación de una función por un programa de una máquina de Turing es como sigue: dado un programa P, con símbolos de cinta A y alfabeto de entrada $E \subset A$, que se detiene para todas las cadenas de entrada, entonces P computa la función $f_P : E^* \dashrightarrow A^*$, donde para cada $x \in E^*$, $f_P(x)$ se define como la cadena obtenida al ejecutar P con la entrada x hasta la detención del programa y formando la cadena con los símbolos de los cuadrados 1 hasta el más a la derecha que sea no blanco.

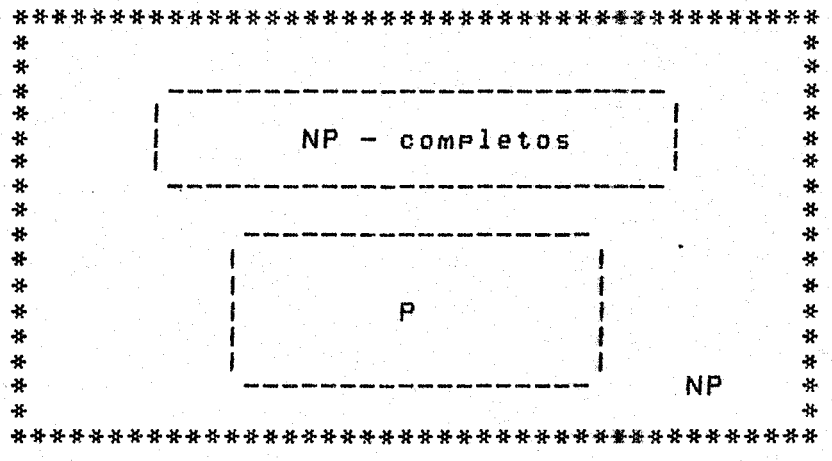
La clase de los problemas NP-completos es muy amplia y contiene a la mayor parte de los problemas clásicos considerados "difíciles". Garey & Johnson (1979) dan una lista de más de trescientos problemas NP-completos diferentes.

Es por ello que es comúnmente admitido que se verifica la siguiente

CONJETURA :

$$P \neq NP$$

Bajo esta hipótesis la relación entre las clases P , NP y la de los problemas NP-completos puede representarse mediante el siguiente esquema:



El primer resultado sobre el caracter NP-completo de un problema lo constituye el Teorema de Cook que afirma que el problema de la satisfactibilidad de una expresión booleana pertenece a tal clase. La demostración de Cook data de 1971.

De entre los numerosos problemas NP-completos nos limitamos a citar seis de ellos que pueden ser considerados básicos, por cuanto que una gran parte de las demostraciones de la NP-completitud de los problemas de secuenciación se basan en la reducibilidad entre éstos y los que a continuación citamos.

Teorema Los siguientes problemas son NP-completos :

- **3-Partición de un conjunto** Dado un conjunto $S = \{1,\ldots,3t\}$ y una familia de subconjuntos S_i de S con $card(S_i) = 3$, para $i=1,\ldots,s$

  ¿ existe una subfamilia de t subconjuntos tales que cada elemento de S está contenido en exactamente un elemento de la subfamilia ?

- **Problema de la Mochila** Dados $t+1$ enteros positivos $z_1,\ldots,z_t,a$

  ¿ existe un subconjunto de índices $S \subset T=\{1,\ldots,t\}$ tal que $\sum_{i \in S} z_i = a$?

- **Circuito dirigido Hamiltoniano** Dado un digrafo $G=(V,A)$

  ¿ contiene G un circuito Hamiltoniano ?

- **Camino dirigido Hamiltoniano** Dado un digrafo $G'=(V',A')$

  ¿ contiene G' un camino Hamiltoniano ?

- **Problema del Viajante** Dado un conjunto finito $C = \{c_1,\ldots,c_t\}$ de "ciudades" con "distancias" $d(c_i,c_j)$ enteras no negativas para cada par de ciudades i,j y un entero positivo z,

  ¿ existe un camino Hamiltoniano sobre C, es decir una ordenación $\langle c_{n(1)},\ldots,c_{n(m)} \rangle$ de C, tal que

$$\left[\sum_{i=1}^{m-1} d(c_{n(i)},c_{n(i+1)}) \right] + d(c_{n(m)},c_{n(1)}) \leq z \ ?$$

- **Subgrafo Completo** Dado un grafo $G = (V,A)$ y $z \in \mathbb{Z}^+$

  ¿ contiene V a K_z (grafo completo de z vértices)?

La demostración de este teorema se debe a Karp (1972) que da una lista de veintiún problemas NP-completos entre los que se encuentran los tres primeros y el último de los enunciados, el tercero y cuarto se demuestran fácilmente viendo que el problema del Circuito dirigido Hamiltoniano se reduce polinomialmente a ellos.

[]

La NP-completitud la hemos desarrollado trabajando con problemas de decisión. Para los problemas generales, entendidos como aquellos en que se busca una solución, hay que recurrir a otro sistema de cómputo : la <<máquina de Turing oráculo>> . Y debe emplearse la <<Turing reducibilidad>> para poder llegar al concepto paralelo de problema <<NP-difícil>> ("NP-hard").

Un <<problema de búsqueda>> B consiste en un conjunto finito C_B de objetos denominados <<configuraciones>> y para cada configuración $c \in C_B$ un conjunto $S_B(c)$ de elementos denominados <<soluciones>>.

Un algoritmo se dice que resuelve un problema de búsqueda B si, dada como entrada una configuración, devuelve como respuesta alguna solución $s \in S_B(c)$, salvo que $S_B(c)$ sea vacío en cuyo caso devuelve la respuesta "no".

Dado un alfabeto finito E , una <<relación de cadena>> es una relación binaria R contenida en el producto cartesiano $E^+ \times E^+$, donde E^+ es el conjunto de todas las cadenas no vacías de elementos de E. Un lenguaje L puede identificarse con la relación de cadena formada por los pares (x,s) donde x es una cadena no vacía perteneciente al lenguaje L y s es cualquier símbolo fijo de E.

La correspondencia entre relaciones de cadenas y problemas de búsqueda está determinada por el esquema de codificación que empleemos, de forma que si tenemos el problema de búsqueda B y el esquema de codificación e , la correspondiente relación de cadena $R[B,e]$ está constituida por los

pares (x,y) siendo $x,y \in E^*$ las codificaciones bajo e de una configuración $c \in C_B$ y una solución $s \in S_B(c)$ respectivamente.

Una reducción Turing de tiempo polinomial de un problema de búsqueda B a otro B', es un algoritmo A que resuelve B por medio de una hipotética subrutina S que resuelve B', tal que si S es un algoritmo de tiempo polinomial para B', entonces A es un algoritmo de tiempo polinomial para B. La formalización de este concepto se basa en la <<máquina de Turing oráculo>> que consiste en una máquina de Turing aumentada con una <<cinta oráculo>> cuyos cuadrados se numeran de forma análoga a la de la primera cinta, y con una <<cabeza-oráculo>> de lectura-escritura que actúa sobre la cinta correspondiente.

Un programa para una máquina de Turing oráculo debe especificar, aparte del alfabeto de símbolos de cinta A con su subconjunto E de símbolos de entrada y el símbolo blanco, un conjunto finito de estados Q que contiene a:

- q_0 estado inicial
- q_P estado de parada
- q_C estado de consulta al oráculo
- q_r estado de reanudación del cómputo.

El programa queda determinado por su función de transición

$$f:(Q - \{q_P, q_C\}) \times A \times E \longrightarrow Q \times A \times E \times \{-1,+1\} \times \{-1,+1\}$$

Los cómputos de una máquina de Turing oráculo comienzan con la entrada x escrita desde el cuadrado 1 al $|x|$ de la primera cinta, estando el resto de ésta y la cinta oráculo en blanco. Cada cabeza resistra un cuadrado de su cinta y el control se encuentra en el estado q_0. Los cómputos se realizan de forma que en cada etapa pueden presentarse uno de los siguientes casos:

- Si el estado es q_P, el cómputo se detiene.

- Si el estado es $q \in Q - \{q_p, q_c\}$, sean s_1 y s_2 los símbolos resistrados en la cinta y en la cinta oráculo respectivamente y sea $f(q,s_1,s_2) = (q', s_1', s_2', \Delta_1, \Delta_2)$. El control pasa de q a q', s_1' se situa en el cuadrado resistrado de la primera cinta y s_2' en la cinta oráculo, y las cabezas avanzan o retroceden un cuadrado, sesún su Δ correspondiente valsa +1 ó -1.

- Si el estado es el q_c, entonces la acción a tomar depende del contenido de la cinta oráculo y de una <<función oráculo>> $s : E^* \longrightarrow E^*$. Sea $y \in E^*$ la cadena que está escrita desde el cuadrado 1 al $|y|$ de la cinta oráculo, el $|y| + 1$ será el primer cuadrado a la derecha del cero que contendrá un símbolo blanco, y sea $z=s(y)$. Entonces la cinta oráculo, en una etapa, cambia para contener la cadena z desde el cuadrado 1 al $|z|$, con blancos en el resto de la cinta, la cabeza de esta cinta se sitúa en el primer cuadrado y el control cambia de q_c al q_r. En esta etapa la primera cinta no sufre ninsuna variación.

La tercera etapa es la que corresponde a la llamada a una subrutina.

Denotamos mediante $\mathbb{P}_s$ el prosrama obtenido al combinar $\mathbb{P}$ con la función oráculo s. Decimos que $\mathbb{P}_s$ es de tiempo polinomial si existe un polinomio p tal que $\mathbb{P}_s$ se detiene en $p(|x|)$ etapas $\forall x \in E^*$.

Si R y R' son relaciones de cadena, una <<reducción Turins de tiempo polinomial>> de R en R' es un prosrama oráculo $\mathbb{P}$ con alfabeto de entrada E, tal que $\forall s: E^* \longrightarrow E^*$ que realiza[*] R' el prosrama $\mathbb{P}_s$ es polinomial y la función $f_{\mathbb{P}}^s$ computada por $\mathbb{P}_s$ realiza R. Esta situación se denota $R \propto_T R'$.

[*] Una función $f: E^* \longrightarrow E^*$ realiza la relación de cadena R si y solo si $\forall x \in E^+$, f(x) es la cadena vacía si $\nexists y \in E^+$ tal que $(x,y) \in R$ o bien es un $y \in E^+$ tal que $(x,y) \in R$. Cuando R es realizable por alsuna función $f_{\mathbb{P}}$ (computada por un prosrama $\mathbb{P}$) se dice que $\mathbb{P}$ resuelve R.

Con lo anterior podemos decir que una relación de cadena R es <<NP-difícil>> si existe algún lenguaje L NP-completo tal que $L \propto_T R$.

Un problema de búsqueda B (bajo el esquema de codificación e) se dice NP-difícil si la relación de cadena R[B,e] es NP-difícil.

Otro elemento básico en la Teoría de la Complejidad es el concepto de <<NP-completitud fuerte>> (también denominada <<NP-completitud unitaria>>) que está estrechamente lisado a la característica de "concisión" que se exise al esquema de codificación. En los problemas en que aparecen números, si la codificación de éstos se realiza en una base distinta de uno, el tamaño de una confisuración dependerá linealmente del número de datos pero solo losarítmicamente sus masnitudes. Si permitimos que el tamaño de la confisuración crezca linealmente con las masnitudes (y ésto ocurre si utilizamos un esquema de codificación unitario) es más frecuente encontrar alsoritmos polinomiales respecto de esta definición de tamaño. Y aquellos problemas que sisuen siendo NP-completos pueden ser considerados los más difíciles entre los de tal clase.

La formalización de la NP-completitud fuerte requiere la definición previa de dos funciones con valores enteros positivos definidas sobre el conjunto de confisuraciones del problema. Para una confisuración dada c, se define:

- Lonsitud[c] = número de símbolos utilizados para describir c bajo algún esquema de codificación.
- Max[c] = masnitud del mayor número de c.

Un alsoritmo que resuelve un problema $\overline{\Pi}$ se denomina <<alsoritmo de tiempo seudopolinomial>> para $\overline{\Pi}$ si su función de complejidad de tiempo está acotada por un polinomio en las variables Lonsitud[c] y Max[c].

En la práctica, para aquellos problemas en los que la

magnitud de los datos no crece excesivamente, (como es el caso de los problemas de secuenciación) la existencia de un algoritmo seudopolinomial hace que este problema sea tratable para la mayoría de los casos de interés real.

Decimos que un problema $\overline{\overline{\Pi}}$ es un problema numérico si no existe ningún polinomio p tal que, para toda configuración c, $Max[c] \leq p(Longitud[c])$.

Se verifica que si $\overline{\overline{\Pi}}$ es NP-completo y no es un problema numérico entonces no puede ser resuelto mediante un algoritmo seudopolinomial salvo que $P = NP$.

Dado un problema $\overline{\overline{\Pi}}$ y un polinomio p, $\overline{\overline{\Pi}}^{p}$ representa el subconjunto de $\overline{\overline{\Pi}}$ obtenido al restringir $\overline{\overline{\Pi}}$ únicamente a aquellas configuraciones c que verifiquen

$$Max[c] \leq p(Longitud[c]) .$$

Si $\overline{\overline{\Pi}} \in NP$ y $\exists p$ / $\overline{\overline{\Pi}}^{p}$ es NP-completo entonces $\overline{\overline{\Pi}}$ se denomina NP-completo fuerte.

Si $\overline{\overline{\Pi}}$ es fuertemente NP-completo ===> $\overline{\overline{\Pi}}$ no puede ser resuelto mediante un algoritmo seudopolinomial salvo que $P = NP$.

Como ejemplos de problemas fuertemente NP-completos tenemos:

3-Partición

Dados $3t + 1$ enteros positivos $z_1, \ldots, z_{3t}$, b con $b/4 < z_K < b/2$ para $K=1,\ldots,3t$,

¿ existe una partición $(S_1, \ldots, S_t)$ del conjunto $\{1,\ldots,3t\}$ tal que $|S_K| = 3$ y

$$\sum_{j \in S_i} z_j = b \quad , \quad j = 1,\ldots,t \quad ?$$

Existencia de solución factible para n|i|r ,<d_i|d

Hasta ahora, la clase más extensa estudiada ha sido la NP a continuación damos una breve reseña de algunas clases con elementos presumiblemente fuera de NP.

Si mediante $\overline{\overline{\Pi}}^{c}$ denotamos el problema de decisión complementario del $\overline{\overline{\Pi}}$ (el obtenido al tomar las respuestas "sí" y "no" intercambiadas), se define

$$\text{co-NP} = \{ \overline{\overline{\Pi}}^{c} \ / \ \overline{\overline{\Pi}} \in \text{NP} \}$$

o en términos de lenguajes

$$\text{co-NP} = \{ E^{*} - L \ / \ L \text{ es un lenguaje sobre } E \text{ y } L \in \text{NP} \}$$

Se verifica :

- Si $\overline{\overline{\Pi}} \in P \Longrightarrow \overline{\overline{\Pi}}^{c} \in P$
- Si $\nexists \overline{\overline{\Pi}}$ NP-completo / $\overline{\overline{\Pi}}^{c} \in$ NP $\Longrightarrow$ NP = co-NP .

Se conjetura que NP ≠ co-NP , por lo que si un problema y su complementario están en NP se suele admitir que este problema no es NP-completo.

Por último, consideramos los problemas que requieren una cantidad de "espacio" acotada por un polinomio en el tamaño de la entrada.

El <<espacio>> utilizado en un cómputo por una máquina de Turing se define como el número de cuadrados distintos visitados por la cabeza de lectura-escritura.

Evidentemente el espacio será siempre menor que el tiempo por lo que los algoritmos de tiempo polinomial también serán de espacio polinomial, aún más, se verifica que todos los problemas de NP son de espacio polinomial.

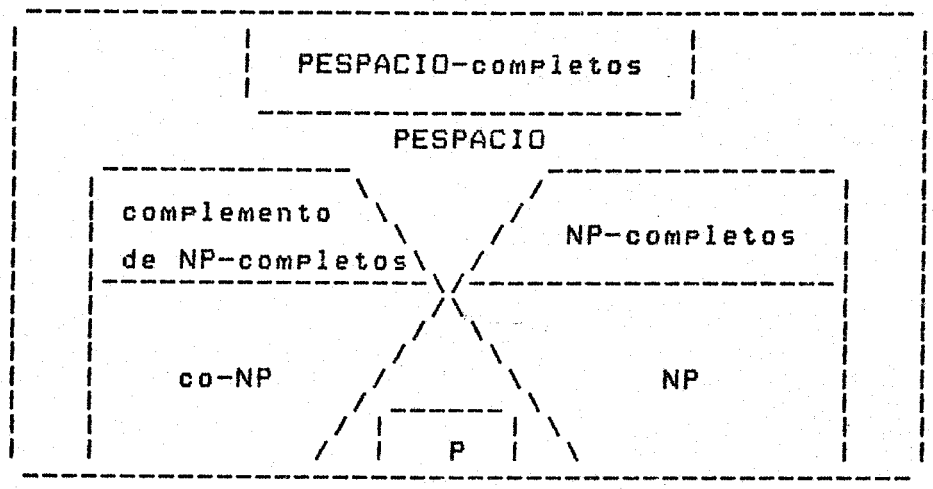
Se define <<PESPACIO>> como la clase de lenguajes reco-

nocibles por programas de espacio polinomialmente acotado que se detienen para todas las entradas.

L se dice <<PESPACIO-completo>> si L ∈ PESPACIO y ∀ L' ∈ PESPACIO L' $\propto$ L.

De aquí se deduce que si L es PESPACIO-completo, L ∈ P si y solo si P = PESPACIO.

Hay que señalar que la verificación de la validez de las conjeturas sobre las que se asienta la Teoría de la Complejidad, principalmente la cuestión sobre si P = NP , probablemente no podrá llevarse a cabo hasta el desarrollo de nuevas técnicas matemáticas. Supuesto que las conjeturas sean ciertas, como indican todos los indicios conocidos, la relación entre las distintas clases de problemas puede representarse mediante el siguiente esquema:



Lo anterior constituye el bagaje indispensable para afrontar el estudio de la complejidad de los problemas de secuenciación, quedan fuera algunos conceptos, por ejemplo los relacionados con clases tales como la #P para los problemas de enumeración [*] y la DLOGESPACIO de lenguajes reconocibles por un programa utilizando un espacio acotado por el logaritmo de la longitud de la entrada (Garey & Johnson 1979).

[*] Un problema de enumeración basado en un problema de bdsqueda B consiste en encontrar, dada una configuración c , cuál es la cardinalidad de $S_B(c)$.

CAPITULO III

Si nuestro problema de secuenciación n|m|c,I|d consiste en la minimización de una medida regular, podemos restringirnos a las secuencias semiactivas. Al ser el conjunto de las mismas finito, es obvio que la primera aproximación a la resolución de este problema es la dada por la enumeración completa. Se tratará de generar las $(n!)^m$ posibles secuencias (si I es vacío), valorando el criterio para cada una de ellas y eligiendo aquella que nos dé el mínimo.

Así, en el caso de ser m = 1, dado un criterio, un procedimiento para su optimización mediante enumeración completa es el siguiente:[*]

```
program enum_completa(input,output);
const n=..; {hay que sustituir ".." por el número de
             trabajos en cada problema particular}
var nn,K,q,opt : integer;
    t,d,w,p,popt : array[1..n] of integer;

{ aquí se sitúa la definición de la función valor }

procedure perm(var K:integer); { procedimiento generador de
                                 las permutaciones}
var v,K_1,i :integer;
begin
 if K=1 then begin
               v:=valor(p);
               if opt < v  then begin
                                  popt:=p;
                                  opt:=v
                                end;
             end
        else begin
               K_1:=K-1;
               perm(K_1);
               for i:=1 to K-1 do
                  begin q:=p[i];
                        p[i]:=p[K];
                        p[K]:=q;
                        K_1:=K-1;
                        perm(K_1)
                  end;
               q:=p[1];
               for i:=1 to K-1 do p[i]:=p[i+1];
               p[K]:=q
             end
end;
begin {comienzo del programa principal}
 nn:=n;
 write('t,d,w?');breaK(output);
 for q:=1 to n do readln(t[q],d[q],w[q]); {lectura de datos}
 for q:=1 to n do p[q]:=q;
 perm(nn);
 write('Optimo =',opt);breaK(output)
end.
```

[*] En el apéndice se introduce el lenguaje empleado para escribir los algoritmos.

El procedimiento recursivo perm genera todas las permutaciones de orden n , valorando el criterio en cada etapa y conservando en cada una de ellas el valor mínimo encontrado hasta ese momento,así como la secuencia al cual corresponde.

La valoración del criterio la realiza la función valor, así en el caso de la minimización de la tardanza ponderada se emplearía la siguiente función:

```
function valor(p : array[1..n] of integer) : integer;
var l,s,i : integer;
    C : array[0..n] of integer;
begin
 C[p[1]]:=t[p[1]];valor:=0;
 for i:=2 to n do
    begin
     l:=p[i];
     C[p[i]]:=C[p[i-1]]+t[p[i]];
     if C[l]>d[l] then valor:=valor+(w[l]*(C[l]-d[l]));
    end
end;
```

El método de enumeración completa es del tipo O(n!) ,lo que provoca su inviabilidad para problemas que no sean de tamaño muy reducido. Basta observar que si suponemos que es posible realizar la valoración del criterio para cada una de las secuencias en un microsegundo, tenemos la siguiente relación entre el tamaño y el tiempo necesario para resolver el problema mediante enumeración completa:

n	tiempo
5	1.2×10^{-4} segundos
10	3.63 "
20	7.71×10^{4} años
30	8.41×10^{16} siglos
40	2.59×10^{32} "
50	9.64×10^{48} "

Por lo que para resolver un problema de tamaño 20 en menos de un minuto sería necesario disponer de un ordenador capaz de evaluar una secuencia en un attosegundo (10^{-18} s.).

Avance que no parece muy probable que se alcance con la tecnología actual.

El no poder recurrir a la enumeración completa nos obliga a utilizar otros métodos. Desde el comienzo del desarrollo de la Teoría de la Secuenciación uno de los primeros caminos intentados ha sido el de la formulación como un problema de programación entera, siendo el criterio más estudiado con esta técnica el de minimización del calendario.

A continuación planteamos un modelo válido para la minimización de la tardanza ponderada. Para ello se utilizará la siguiente notación (basada en la desarrollada en el apartado I.1) :

T cota superior de C_{max}

q_{ijh} cantidad de recurso h requerido por O_{ij}

q_{ht} cantidad de recurso h existente en el instante t ($q_{ht} = q_h \ \forall t$ si no estamos en el caso "rec var")

$$l_{ij} = r_i + \sum_{\substack{O_{iK} < O_{ij}}} t_{iK}$$ cota inferior de S_{ij}

$$u_{ij} = T - \sum_{\substack{O_{ij} < O_{iK}}} t_{iK}$$ cota superior de C_{ij} .

Definimos dos conjunto de variables:

$X_{ijt} = 1$ si $C_{ij} = t$ $\quad i = 1,\ldots,n$ $\quad j = 1,\ldots,n_i$ $\quad t = 1,\ldots,T$

$= 0$ en otro caso.

$X_{it} = 1$ si $C_i < t$

$= 0$ en otro caso.

Con estas definiciones, se verificará que el trabajo J_i es tardío si X_{it} vale cero siendo $t > d_i$, por lo que la tardanza total vendrá dada por:

$$\sum_{i=1}^{n} \sum_{t=d_i+1}^{T} (1 - X_{it})$$

Y la minimización de la tardanza ponderada será equivalente a la maximización de:

$$\sum_{i=1}^{n} w_i \sum_{t=d_i+1}^{T} X_{it}$$

Las restricciones del modelo dependerán fundamentalmente del conjunto I que indica las hipótesis relajadas con respecto al modelo básico de secuenciación. Para el caso de ser $I = \{ r_i > 0, prec, rec\ var \}$ tendremos las siguientes:

1.- Completación de las operaciones

$$\sum_{t=1}^{u_{ij}} X_{ijt} = 1 \qquad \begin{matrix} i = 1,\ldots,n \\ j = 1,\ldots,n_i \end{matrix}$$

2.- Completación de los trabajos

$$X_{it} - \frac{i}{n_i} \sum_{j=1}^{n_i} \sum_{K=1}^{t-1} X_{ijK} \leq 0$$

con $t = 1_{if} + t_{if}, \ldots, T$ siendo O_{if} la operación terminal del trabajo J_i

3.- Recursos

En cada instante no podemos consumir más recursos de los existentes, por tanto

$$\sum_{i=1}^{n} \sum_{j=1}^{n_i} \sum_{K=t}^{t+t_{ij}-1} q_{ijh} X_{ijK} \leq q_{ht}$$

con $h = 1,\ldots,s$

y $t = 0,\ldots,T$ (suponemos que el primer trabajo comienza a ser procesado en el instante inicial).

4.- Restricciones de precedencia

$$\text{Si } O_{ij} < O_{iK} \quad ===> \quad C_{ij} + t_{ij} \leq C_{iK}$$

y

$$C_{ij} = \sum_{t=1}^{u_{ij}} t \, X_{ijt}$$

deben verificarse, por tanto, las siguientes restricciones:

$$\sum_{t=1}^{u_{iK}} t \, X_{iKt} - \sum_{t=1}^{u_{ij}} t \, X_{ijt} \geq t_{ij}$$

Estas restricciones corresponden a las relaciones de precedencia entre operaciones. Para las relaciones entre trabajos basta imponer que la última operación del trabajo predecesor se complete antes de comenzar la primera del sucesor.

El modelo planteado consta, en principio, de NT variables del primer tipo (con $N = \sum_{i} n_i$), y nt del segundo.

El número de restricciones es:

N del tipo 1.
n del tipo 2.
sT del tipo 3.

El número de las del tipo 4 dependerá de la densidad de la red que describe las relaciones de precedencia, como máximo habrá

$$\sum_{i=1}^{n} \frac{n_i^2 - n_i}{2}$$ para las operaciones

y $\frac{n^2 - n}{2}$ para los trabajos.

El número de variables es de hecho más pequeño, debido a que muchos valores están prefijados, así, por ejemplo:

$$X_{ijt} = 0 \quad \text{para} \quad t < l_{ij} \quad o \quad t > u_{ij} \quad .$$

Sin embargo, el modelo resultante una vez eliminadas todas estas variables (y las posibles restricciones redundantes), sigue siendo demasiado grande para problemas de tamaño medio, dependiendo en gran parte la efectividad del método de la magnitud de los datos implicados en el mismo, esta observación puede hacernos pensar en la posibilidad de que este problema sea seudopolinomial, posteriormente veremos que, en el modelo general, no es éste el caso.

Para el caso de ser el conjunto R de recursos vacío, Manne (1960) plantea un modelo de programación entera, en el que las variables a determinar X_{ij} se definen como el tiempo de comienzo del trabajo i en la máquina j, y para introducir las relaciones de precedencia se recurre a unas variables $X_{ij\,qk}$ del tipo 0-1, que toman el valor uno si $O_{ij} < O_{qk}$ en la máquina $\#(O_{ij})$ (supuesto que $\#(O_{ij}) = \#(O_{qk})$).

También para el caso de ser R vacío, Balas (1969) plantea el problema de la minimización del calendario como un problema de camino crítico sobre el grafo dado por la representación derivada del concepto de grafo disyuntivo. Aunque las experiencias computacionales obtenidas para los modelos de Manne y Balas muestran que no son demasiado efectivos, una cierta mejora ha sido lograda por Land, Laporte & Miliotis (1978) que muestran que ambas formulaciones están estrechamente relacionadas, de forma que pasando al dual el problema de programación lineal entera de Manne se obtiene el modelo de Balas. Y si se utiliza para la resolución del problema de PLE un algoritmo de ramificación y acotación, el sondeo de

las ramas puede realizarse más rápidamente utilizando un algoritmo "out-of-kilter" para los problemas de flujo.

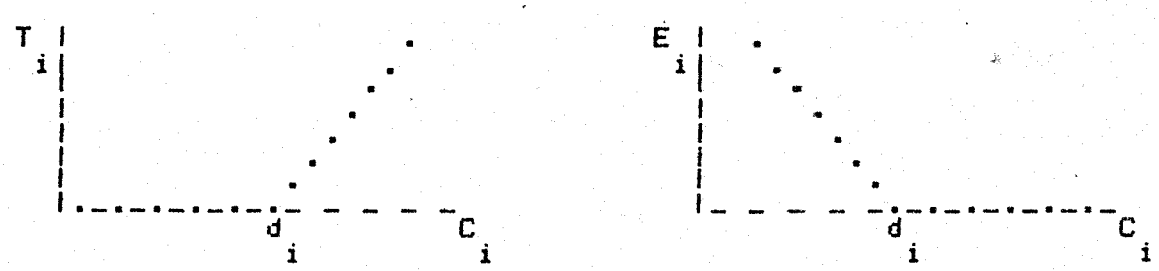
En resumen, podemos afirmar que aunque se han obtenido algunos resultados mediante el planteamiento por programación 0-1 (principalmente para el calendario), es preciso recurrir a técnicas más específicas que sean capaces de aprovechar la estructura particular de los problemas de secuenciación.

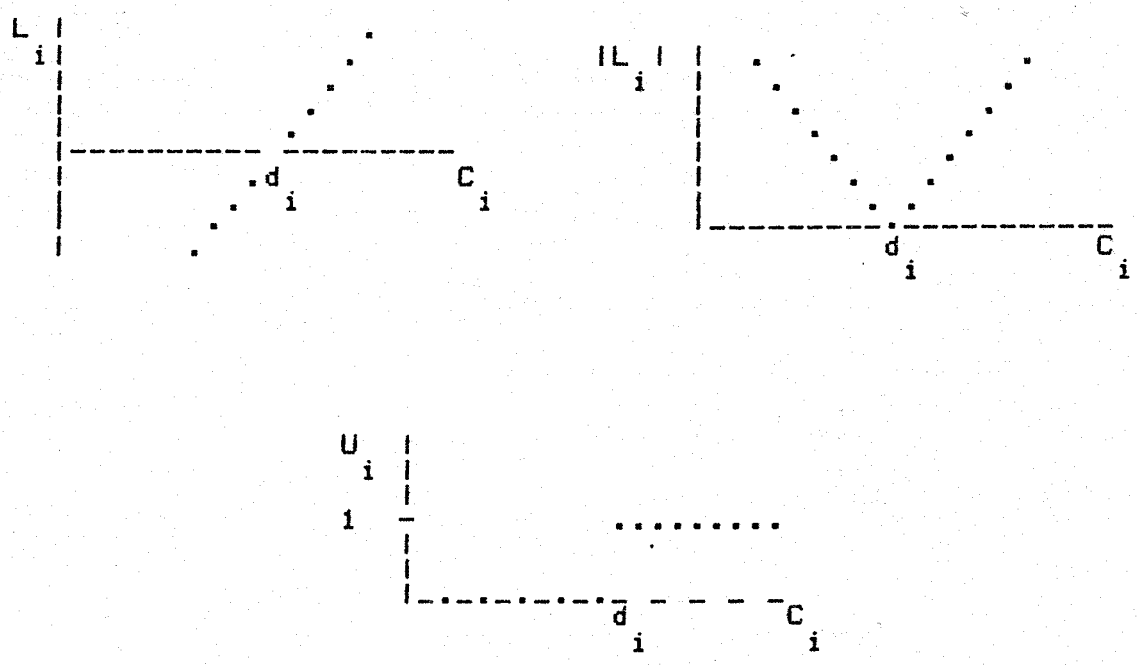
En lo que sigue, centramos nuestra atención sobre los criterios que implican fechas de entrega, el interés de estos criterios es tanto práctico como teórico, puesto que uno de los problemas reales de secuenciación más frecuentes es el de lograr que los trabajos se completen antes de sus fechas de entrega. La conveniencia de utilizar uno u otro criterio viene dada en función de las características de cada problema, así, por ejemplo, cuando no se obtiene ningún beneficio por disminuir las fechas de completación por debajo de las fechas de entrega y además la penalización por ser el trabajo tardío es proporcional al retraso, la cuantificación más natural del objetivo de completar los trabajos a tiempo es la dada por el criterio de la minimización de la suma ponderada de las tardanzas:

$$\sum w_i T_i \quad .$$

Además de la tardanza, las medidas más utilizadas cuando existen fechas de entrega son las del desfase (L_i), adelanto (E_i), desfase absoluto ($|L_i|$) e índice de tardanza (U_i).

Gráficamente estas medidas pueden ser representadas de la siguiente forma:





En los próximos apartados se estudian los principales resultados conocidos y los obtenidos por nosotros para los criterios basados en estas medidas.

III.1 Minimización del máximo costo

Para el caso de una sola máquina (m=1), el primer resultado conocido corresponde a la minimización del desfase máximo (y por tanto a la minimización de T_{max} y a la maximización de E_{max}) y consiste en que L_{max} alcanza el mínimo para la secuencia obtenida al ordenar los trabajos en orden no decreciente de fechas de entresa (secuencia EDD, "Earliest Due Date"). Este resultado es debido a Jackson (1955) y la demostración puede verse como un caso particular del arsumento de intercambio de pares adyacentes de Smith (1956).

El alsoritmo de Jackson, como todos aquellos que requieren una simple ordenación de n números, es del tipo $O(n \log_2 n)$.

Una implementación inspirada en la versión recursiva del método rápido de ordenación de Hoare (1971) es la sisuiente:

Suponemos que la información sobre los trabajos se almacena en un vector de resistros, en el que cada componente contiene el tiempo de procesamiento y la fecha de entresa de un trabajo, es decir,

```
type  trabajo = record
                  tiempo,fecha : integer
                end

Jobs : array[1..n] of trabajo
```

Con estos elementos podemos construir el procedimiento:

```
procedure Jackson;
begin
  ord_EDD(1,n)
end;
```

donde ordJackson es la subrutina dada por:

```
procedure ord_EDD(iz,de : integer);
var i,J : integer;
    x,w : trabajo;

begin {procedimiento de ordenación}
   i:=iz; J:=de;
   x:=Jobs[(iz+de) div 2];
   repeat
      while Jobs[i].fecha<x.fecha do i:=i+1;
      while x.fecha<Jobs[J].fecha do J:=J+1;
```

```
        if i<=J then
                begin
                  w:=Jobs[i];
                  Jobs[i]:=Jobs[J];
                  Jobs[J]:=w;
                  i:=i+1;
                  J:=J-1;
                end
    until i>J;
    if iz<J then ord_EDD(iz,J);
    if i<de then ord_EDD(i,de)
end;
```

Por simplicidad en la escritura de los procedimientos se utilizan, en vez del vector de registros Jobs, dos vectores t y d de tiempos de procesamiento y fechas de entrega. Por lo que, siempre que otro procedimiento utilice como subrutina el procedimiento Jackson, suponemos que éste ha sido modificado añadiéndole al final la instrucción:

```
for i:=1 to n do begin
                   t[i]:=Jobs[i].tiempo;
                   d[i]:=Jobs[i].fecha
                 end;
```

El algoritmo de Jackson es la base para la generalización al caso $n|1|t_{max}$ de Vickson (1980).

En nuestra implementación de este algoritmo utilizamos la siguiente notación:

- t_i tiempo de procesamiento inicial de J_i.
- l_i tiempo mínimo de procesamiento.
- c_i costo en que se incurre al disminuir en una unidad el tiempo de procesamiento.
- x_i variable que nos da el número de unidades en que se disminuye t_i.
- $m_i = t_i - l_i$ valor máximo de x_i.
- $p_i = t_i - x_i$ valor actual del tiempo de procesamiento.

Estas cantidades se almacenan en vectores de n componentes, es decir,

```
t,l,c,x,m,p : array[1..n] of integer
```

donde m se construye con el ciclo

```
for i:=1 to n do m[i]:=t[i]-l[i];
```

Las tardanzas las obtiene el procedimiento tard

```
procedure tard;
var T,C : array[1..n] of integer; { T tardanzas, C tiempos
                                    de completación}
begin
   Jackson; {ordenación de los trabajos según EDD}
   C[1]:=t[1];
   for i:=2 to n do
          begin
             C[i]:=c[i-1]+t[i];
             if C[i]>d[i] then T[i]:=C[i]-d[i]
                          else T[i]:=0
          end
end;
```

Aunque T y C se han definido como variables locales y no aparecen como parámetros de salida del procedimiento, trabajeremos como si fuesen globales, de forma que podremos utilizar directamente sus valores en cualquier otro procedimiento. Esta observación es válida para todas las variables locales que utilicemos en los procedimientos que a continuación se desarrollan.

La tardanza máxima es calculada por Tmax , que almacena su valor en la variable entera Tm y el menor índice en que ésta se alcanza en la N .

```
procedure Tmax;
var Tm,N : integer;
begin
Tm:=0; N:=1;   {inicializacion}
for i:=1 to n do
        if Tm<T[i] then begin
                           Tm:=T[i];
                           N:=i
                        end
end;
```

En caso de que queramos obtener el máximo entre los K primeros componentes de un vector de dimensión n , se emplea la función max definida:

```
function max(T:array[1..n],k):integer;
begin
max:=T[1];
for i:=2 to K do
        if T[i]>max then max:=T[i]
end;
```

La condición de optimalidad de una solución es testada por la función booleana Copt , de forma que si la condición se verifica toma el valor "true" .

```
function  Copt:boolean;
var c1,c2,c3:boolean;
begin
 c1:= Tm=0; {solución óptima por ser la tardanza nula}
 c2:= true;
 c3:= true;
 for i:=1 to n do
       begin
          c2:= c2 and (c[i]>1);     {solución óptima por no
                                     poderse disminuir Tmax
                                     sin aumentar el coste}
          c3:= c3 and (p[i]=l[i])   {solución óptima por no
                                     ser posible acortar más
                                     los tiempos}
       end;
 Copt:= c1 or c2 or c3
end;
```

El cálculo del índice del trabajo con menor fecha de entrega,de entre los más baratos de acortar y anteriores al N, lo realiza Jota .

```
      function Jota:inteser;
      var c:inteser;
      begin
      c:=maxint;
      for i:=1 to N do
                  if (p[i]>1[i]) and (c>c[i]) then
                                                 begin
                                                   Jota:=i;
                                                   c:=c[i]
                                                 end
      end;
```

Estos procedimientos y funciones auxiliares nos permiten construir el procedimiento n_1_tvar_Tmax.

```
procedure n_1_tvar_Tmax;
var jp,xjp,i:inteser;
begin
   Tard;
   Tmax;
   while not Copt do
         begin
           jp:=Jota;
           xjp:=Tm-max(T,jp);
           if m[jp]<xjp then   {podemos disminuir todo lo po-
                               sible el trabajo  jp  sin que
                               ningún otro  se hasa  máxima-
                               mente tardío}
                          begin
                            p[jp]:=1[jp];
                            Tm:=Tm-m[jp];
                            for i:=jp to n do
                                  if T[i]>m[jp] then
                                                T[i]:=T[i]-m[jp]
                                                else
                                                T[i]:=0
                          end
```

```
                    else   {el trabajo JP se puede dis-
                            minuir como máximo xJP unida-
                            des antes de hacer que otro
                            trabajo sea máximamente tar-
                            dío}
                      begin
                         p[JP]:=t[JP]-xJP;
                         Tm:=Tm-xJP;
                         for i:=JP to n do
                                if T[i]>xJP then
                                              T[i]:=T[i]-xJP
                                else
                                              T[i]:=0;
                         Tmax
                      end
   end  {de while}
end
```

Teorema — El procedimiento n_1_tvar_Tmax minimiza la tardanza máxima para el caso de tiempos de procesamiento controlables.

Demostración

El procedimiento en cada etapa va disminuyendo el valor de T_{max} en la forma de menor coste posible, el que este método nos da el óptimo puede verse planteando el problema de una forma equivalente por programación lineal;

$$\text{Min} \quad \sum_{j=1}^{n} c_j x_j + T_{max}$$

Sujeto a

$$T_{max} \geq \sum_{i=1}^{j} (t_i - x_i) - d_j$$

$$0 \leq x_j \leq m_j$$

y $T_{max} \geq 0$

para $j = 1,2,\ldots,N$

Como $L_j = \sum_{i=1}^{j} t_j - d_j$ las restricciones se podrán escribir de la forma

$$\sum_{i=1}^{j} x_i + T_{max} \geq L_j$$

Con esta formulación serán redundantes las restricciones

en que aparezcan desfases negativos, o bien aquellas j tales que $\exists L_K > L_j$ con $K<j$.

El modelo puede entonces formularse como

$$\sum_{i=1}^{j_K} x_i + T_{max} \geq L_{j_K}$$

con $K=1,2,\ldots,r$

donde $0 < L_{j_1} < \ldots < L_{j_r}$ y $1 \leq j_1 < j_2 < \ldots < j_r$ si $r>1$

Este último modelo corresponde a un problema de producción con demandas no negativas:

$$r_K = L_{j_K} - L_{j_{K-1}} \quad \text{para el periodo } j_K ,$$

con capacidad de producción limitada por m_j en el periodo j, y un inventario inicial igual a T_{max} .

El costo unitario de producción es c_j y no existe costo de almacenamiento. Para este problema es obvio que la solución óptima se obtiene seleccionando el periodo de menor costo en el que es posible satisfacer la demanda y en tal periodo producir toda la capacidad, o la cantidad necesaria para satisfacer la demanda en ese momento y en los siguientes periodos.

Este método aplicado reiteradamente es el mismo que realiza nuestro procedimiento.

[]

El algoritmo es del tipo $O(n^2)$. Sin embargo, en el caso de que el tiempo de procesamiento solo pueda tomar, para cada trabajo i, los valores t_i y l_i (este último con un costo c_i m_i) el problema deja de ser polinomial para hacerse NP-difícil, lo que puede verse al observar que se puede plantear como:

$$\text{Min} \quad \sum_{j=1}^{n} (c_j m_j) y_j + T_{max}$$

Sujeto a

$$\sum_{i=1}^{j} m_i y_i + T_{max} \geq L_{j_K} \qquad K=1,\ldots,r$$

$$T_{max} \geq 0$$

$$y_i = 0,1$$

y se puede reducir polinomialmente la versión cero-uno del problema de la mochila a este problema de donde se deduce el carácter NP-difícil del mismo.

Para el problema $n|1|c_{max}$, siendo $c_i(t)$ una función arbitraria no decreciente en t, Lawler (1973) da un algoritmo del tipo $O(n^2)$.

Sea p el vector de posición de los trabajos en la secuencia (p[i] indica el orden en que se procesa J_i), el siguiente procedimiento nos da el p que minimiza c_{max}, donde los costes $c_i(t)$ vienen dados por una función c(i,t):

```
procedure  n_1_cmax;
var K,C,cm : integer;
    p:array[1..n] of integer;
    S:set of 1..n;
begin
    S:=[1..n];
    K:=n;
    for i:=1 to n do C:=C+t[i];
    cm:=maxint;
    while  S<>[] do
         begin
           for l:=1 to n do          {elección del trabajo con
                                      menor costo}
               if (l in S) and (c(l,C)<cm) then
                                          begin
                                             cm:=c(l,C);
                                             i:=l
                                          end;
           p[K]:=i;
           K:=K-1;
           C:=C-t[i];
           S:=S-[i]
         end;
end;
```

Este procedimiento tiene el inconveniente de que la mayoría de las implementaciones del PASCAL (y de cualquier otro lenguaje dotado de la estructura conjunto) limitan el tamaño de los conjuntos a la longitud de la palabra, por representarse cada elemento del tipo base por un bit (significando cero ausencia y uno presencia) por lo que en un PDP 11-23 se admitirán como máximo conjuntos con n=16 y en un UNIVAC 1108 con n=36. Sin embargo,emplearemos siempre que sea posible la estructura conjunto por la mayor intelisibilidad de los procedimientos resultantes. Para tamaños mayores podemos recurrir a esta otra implementación:

Utilizamos el vector P de la misma forma que antes, salvo que inicialmente todas sus componentes son cero.

```
procedure  n_1_cmax(2)
var  K,C,cm,P:inteser;
     P:array[1..n] of inteser;

besin
   P:=0;
   for i:=1 to n do
              besin
                 C:=C+t[i];
                 P[i]:=0
              end;
   K:=n;
   cm:=maxint;
   while P=0 do
     besin
       for l:=1 to n do
          if (P[l]=0) and (c(l,C)<cm) then
                                          besin
                                            cm:=c(l,C);
                                            i:=l
                                          end;
       P[K]:=i;
       K:=K-1;
       C:=C-t[i];
       For i:=1 to n do  P:=P*P[i];
     end;
end;
```

Teorema El procedimiento n_1_cmax minimiza c_{max}.

Demostración

Si es elesido J_j en vez de J_i con

$$c_j \left(\sum_{h \in S} t_h \right) > \min_{l \in S} c_l \left(\sum_{h \in S} t_h \right)$$

entonces intercambiando J_i con los trabajos que le siguen

hasta incluir el J_j, no incrementaremos c_{max} puesto que

$$c_i \left(\sum_{h \in S} t_h \right) < c_j \left(\sum_{h \in S} t_h \right)$$

y

$$c_j \left(\sum_{\substack{h \in S \\ h \neq i}} t_h \right) \leq c_j \left(\sum_{h \in S} t_h \right)$$

[]

Evidentemente la regla de Jackson es un caso particular de este teorema.

El hecho de introducir tiempos de preparación hace al problema de la minimización del máximo costo mucho más difícil, así Rinnooy Kan (1976) demuestra que $n|1|r > 0|L_{max}$ es NP-completo viendo que a él se puede reducir el problema de la mochila.

El algoritmo de Lawler es de hecho más general puesto que es aplicable al problema $n|1|prec|c_{max}$, basta con que en cada etapa se elija el trabajo con menor coste de entre los que no tienen sucesores. La implementación de este procedimiento dependerá fundamentalmente del sistema elegido para representar las relaciones de precedencia, así si para cada trabajo J_i tenemos construido el conjunto A[i] formado por los índices de los trabajos sucesores del i, bastará sustituir la expresión booleana

```
(l in S) and (c(l,C) < cm)
```

por

```
(l in S) and (c(l,C) < cm) and (A[l]*S = [])
```

.

El resultado más general relativo a la minimización del máximo costo es debido a Monma (1980). En los modelos anteriores cada trabajo requiere una cantidad de tiempo t_i para ser procesado, una generalización es suponer que cada trabajo J_i consume una cantidad de recursos q_i que no tiene

necesariamente que ser no negativa sino que puede ser menor que cero en caso de que el trabajo J_i, en vez de consumir produzca $|q_i|$ unidades del recurso.

Para cualquier secuencia P,

$$Q_i^P = \sum_{j=1}^{i-1} q_{P(j)}$$

es el nivel de recursos consumidos hasta el momento en que empieza a ser procesado J_i (suponemos $Q_1^P = 0$, $\forall P$).

Con cada trabajo hay asociada una función no decreciente

$h_i(x)$ = costo en que incurre J_i si empieza a ser procesado cuando el nivel de recursos es x.

El problema denotado h_{max} es encontrar una secuencia que minimice el máximo costo por trabajo.

Si $q_i \geq 0$ $\forall i$, el recurso puede ser considerado el tiempo y estamos en el caso c_{max} que puede ser resuelto por el algoritmo de Lawler incluso si existen relaciones de precedencia.

Si $q_i \leq 0$ $\forall i$, denotamos el problema mediante $\bar{h}_{max}$.

Teorema: c_{max} y $\bar{h}_{max}$ son equivalentes[*].

Demostración

Dados los $t_i \geq 0$ y $c_i(.)$ del problema c_{max} construimos

$$q_i = -t_i \geq 0$$

y

$$h'_i(x) = c_i\left(x + \sum_{j=1}^{n} t_j - t_i\right) \quad \text{para } \bar{h}_{max}.$$

[*] Los criterios f y g se dicen equivalentes si $f \propto g$ y $g \propto f$.

El costo para cualquier secuencia p_i de $\bar{h}_{max}$ es igual al costo de la secuencia inversa p para c_{max}, por tanto $c_{max} \propto \bar{h}_{max}$.

Recíprocamente, dados los q_i y las $h_i(.)$ para el problema $\bar{h}_{max}$ se definen los t_i como sus opuestos y

$$c_i(x) = h_i\left(x + \sum_{j=1}^{n} q_j - q_i\right) \text{ para } c_{max}.$$

Y con el mismo razonamiento $\bar{h}_{max} \propto c_{max}$.

[]

Definimos los conjuntos de trabajos "consumidores", "productores" y "nulos"

$JC=\{i / q_i > 0\}$ $JP=\{i / q_i < 0\}$ $JN=\{i / q_i = 0\}$.

Utilizando los vectores auxiliares s,t y p de n componentes y la función h(i,x) ,podemos construir el siguiente procedimiento:

```
procedure  n_1_hmax;
var  q,s,t,p:array[1..n] of integer;
     JP,JCN:set of 1..n;
     is,it:integer;
begin
   JP:=[];JCN:=[];
   y:=0;
   for i:=1 to n do      {construcción de los conjuntos}
      begin
        if q[i]>0 then JP:=JP+[i]
                  else JCN:=JCN+[i];
        y:=y+q[i]
      end;
   is:=0;it:=n+1;
   x:=0;
   hp:=maxint ; hcn:=maxint;

   while  JP<>[] do     {ordenación hacia detrás}
      begin
        for i:=1 to n do
           if (i in JP) and (h(i,x)<hm) then
                                          begin
                                            hp:=h(i,x);
                                            k:=i
                                          end;
           is:=is+1;
           s(is):=k;
           x:=x+q[k];
           JP:=JP-[k]
      end;

   while JCN<>[] do   {ordenación hacia adelante}
      begin
        for i:=1 to n do
```

```
            if (i in JCN) and (h(i,y)<hcm) then
                                                  begin
                                                    hcn:=h(i,J);
                                                    K:=i
                                                  end;
          it:=it-1;
          t[it]:=K;
          y:=y-q[K];
          JCN:=JCN-[K]
        end;

{construcción de la secuencia óptima}
    for i:=1 to is do p[i]:=s[i];
    for i:=it to n do p[i]:=t[i]
end;
```

Teorema — La secuencia p obtenida por n_1_hmax es óptima.

Demostración

Primeramente, la secuencia óptima debe estar constituida por los trabajos en JP seguidos por los de JN, ya que si $J \in JP \cup JN$ e $i \in JC \cup JN$ con $J_i < J_j$, al ser $q_i \geq 0$ y $q_j < 0$, si intercambiamos los trabajos i y J los niveles de recursos consumidos desde el comienzo del procesamiento del trabajo i al final del J disminuirán y al ser h no decreciente el costo no aumentará.

Para un nivel de recursos consumidos x, de entre los trabajos en JP U JN debe procesarse primero el de menor h(x). Ya que si en una secuencia se tiene (...i..J...) con $J \in JP \cup JN$ y $h_j(x) < h_i(x)$ intercalando el J delante del i, la nueva secuencia (...Ji...) tendrá menor coste por tener todos los trabajos, a excepción del J, niveles de recursos no superiores y ser $h_j(x) < h_i(i)$.

Para los trabajos consumidores o nulos el razonamiento es análogo al empleado en la demostración de que n_1_cmax minimiza c_{max}.

[]

Baker y Nuttle (1980) estudian la generalización al caso de existir un único recurso siendo su disponibilidad variable.

En el caso n|l|rec| var|c se supone la existencia de n trabajos con unas demandas q_i^{max}, i=1,..,n de recursos si t representa el tiempo, r(t) es la cantidad de recursos disponibles en el tiempo t y R(t) es la cantidad de recursos acumulada hasta t,

$$R(t) = \sum_{K=0}^{t} r(K)$$

Cuando la existencia de recursos es constante a lo largo del tiempo, existe una correspondencia lineal entre tiempo y recursos consumidos. Sin embargo,al variar la disponibilidad se pierde la linealidad de la correspondencia, pero sigue siendo posible calcular el tiempo que corresponde a cada nivel de recursos consumidos, ya que dado un nivel de recursos x , el t correspondiente será aquel que verifique $R(t) \geq x$ y $R(t-1) < x$.

Siguiendo la línea desarrollada para n_1_cmax podemos construir un procedimiento para la resolución de este problema.

La primera cuestión es decidir sobre la estructura a emplear para contener la información referente a la disponibilidad de recursos, hemos optado por utilizar un vector R[t] para la disponibilidad acumulada, viniendo dado entonces r[t] por la diferencia R[t]-R[t-1], la dificultad estriba en el campo de variación del índice t ,puesto que habrá que determinar el valor de C_{max} (calendario) tal que $R[C_{max}]$ es igual al total de recursos demandados por los n trabajos

Al depender C_{max} de

$$S_q = \sum_{i=1}^{n} q_i$$

el dimensionamiento del vector R variará con las configuraciones particulares del problema aunque n sea constante.

Supuesto que $r[t] \geq 1 \quad \forall t$, una cota superior de C_{max}

viene dada por S_q y esta constante se utilizará como dimensión del vector R. Un planteamiento alternativo consiste en utilizar un fichero o una lista lineal para almacenar los valores de R, ésto nos producirá un ahorro de memoria, que dependerá de la magnitud de los valores de r, a costa de un aumento en los tiempos de cálculo debido a la necesidad de emplear técnicas menos eficientes de búsqueda.

Por tanto , definimos:

```
R : array [0..Sq]  of integer
```

.

Dado un nivel de recursos x , la función tiempo obtiene el valor de t correspondiente.

```
function  tiempo(x:1..Sq) : 1..Sq;
var i,j,k :integer;
begin
   i:=1;j:=Sq;
   repeat
      k:=(i+j) div 2;
      if x > R[k] then i:=k+1
                  else j:=k-1;
   until (R[k]>=x) and (R[k-1]<x)
   tiempo:=k
end;
```

Los costos $c_i(t)$,al igual que el caso de no existir recursos, vienen dados por una función c(i,t) .

Con estas funciones auxiliares podemos construir el siguiente procedimiento:

```
procedure  n_1_resivar_cmax;
var k,Q,cm,T :integer;
    q,p : array[1..n] of integer;
    S : set of 1..n;
begin
    S:=[1..n];
    k:=n;
    cm:=maxint;
    Q:=Sq;
    while  S<>[] do
       begin
          T:=tiempo(Q);
          for l:=1 to n  do
             if (l in S) and (c(l,T)<cm) then
                                              begin
                                                cm:=c(l,T);
                                                i:=l
                                              end;
          p[k]:=i;
          k:=k-1;
          Q:=Q-q[i];
          S:=S-[i]
       end
 end;
```

El procedimiento n_1_cmax es del tipo $O(n^2)$, el n_1_reclvar_cmax requiere además calcular el tiempo T n veces. Cada uno de estos cálculos lo realiza la función tiempo, la cual utiliza el método de búsqueda binaria por lo que requiere a lo más

$$\log_2 \sum_{i=1}^{n} q_i$$

etapas, por lo que el número total de etapas será:

$$n^2 + n \log_2 \sum_{i=1}^{n} q_i$$

Esto significa que el algoritmo no es polinomial sino seudopolinomial, por lo que su eficiencia dependerá de la magnitud del total de recursos precisos para el procesamiento de los n trabajos.

Con esto podemos afirmar que el problema no es fuertemente NP-difícil. Sin embargo al aumentar el número de procesadores la situación cambia ya que $n|3|P,res1,t_j=1|C_{max}$ es NP-difícil en sentido fuerte (Garey & Johnson, 1975).

III.2 Minimización del costo total

Mientras que los problemas de minimización del máximo costo son, como hemos visto, frecuentemente polinomiales, los relativos al costo total cuando existen fechas de entrega son, en general, más difíciles y de hecho se establecerá el carácter NP-completo de algunos de ellos incluso en el sentido fuerte.

Una excepción la constituye $n|1|\sum U_i$ para el cual existe un algoritmo polinomial (concretamente $O(n \log n)$) debido a Moore & Hodgson (1968) y simplificado por Sturm (1970). Sin embargo, la introducción de relaciones de precedencia hace este problema mucho más difícil, incluso si estas relaciones son de un tipo tan simple como las de cadena y los tiempos de procesamiento son unitarios.

Teorema $n|1|t_i=1, cadena|\sum U_i$ es NP-difícil.

Demostración

Bastará ver que existe un problema de decisión NP-completo que se reduce polinomialmente a la versión de decisión de $n|1|t_i=1, cadena|\sum U_i$.

Este problema será el de la 3-Partición de un conjunto, ya que para cualquier configuración del mismo podemos construir una para el de minimización del número de trabajos tardíos con tiempos de procesamiento unitarios y relación de precedencia tipo cadena.

Al ser la relación del tipo cadena podemos sustituir un trabajo con tiempo de procesamiento t_i por una cadena con t_i trabajos con tiempos unitarios, por ello dada una configuración del problema de la 3-Partición del conjunto nos basta obtener una configuración del $n|1|cadena|\sum U_i$ de la siguiente forma:

- Por cada ocurrencia de un elemento J en un subconjunto S_i formamos un trabajo J_{ij} con $t_{ij} = sJ$ y $d_{ij} = t + \frac{1}{2} sJ(J+1)$ con $i = 1,\ldots,s$ $J \in S_i$

- Por cada S_i tomamos un trabajo J_i con $t_i = 1$ y $d_i = d$ con

$$d = s + s \sum_{i=1}^{s} \sum_{J \in S_i} J$$

Existirán, por tanto, $3s + s = 4s$ trabajos.

- Para cada $S_i = \{i,J,K\}$ con $J < K < l$ se verifica la relación de cadena $J_i << J_{ij} << J_{iK} << J_{il}$.

Con esta construcción, Lenstra y Rinnooy Kan (1980) demuestran que el problema de la 3-Partición del conjunto tiene solución si y solo si existe una secuencia factible con valor $\sum U_i \leq 3(s-t)$.

[]

El caso ponderado $n|1|\sum w_i U_i$ es NP-completo, aún sin existir relaciones de precedencia (Karp, 1972).

La complejidad de $n|1|\sum T_i$ es una cuestión que permanece abierta, aunque se conjetura su NP-completitud. Un algoritmo seudopolinomial para este problema es debido a Lawler (1977).

La introducción de relaciones de precedencia o de pesos hace de nuevo el problema más difícil.

Teorema $n|1|\sum w_i T_i$ y $n|1|t_i=1,cadena|\sum w_i T_i$ son fuertemente NP-difíciles.

Demostración

Trabajaremos con las versiones de decisión de ambos problemas y tendremos que establecer por tanto su carácter fuertemente NP-completo.

Primeramente, una configuración de $n|1|\sum w_i T_i$ puede transformarse polinomialmente en una del problema con tiempos unitarios y relación de precedencia tipo cadena, basta sustituir cada trabajo J_i por una cadena

$$J_{i1} << \ldots << J_{it_i}$$

de t_i trabajos con tiempos unitarios de forma que los $(t_i - 1)$ primeros tengan pesos nulos y el t_i peso igual a w_i, mientras que todos tienen la misma fecha de entresa d_i.

Podemos ahora restringirnos al caso de la tardanza ponderada con tiempos de procesamiento arbitrarios y sin relaciones de precedencia.

Dada una configuración del problema de la 3-Partición (que es fuertemente NP-completo) construimos:

- Para cada $J \in \{1,\ldots,3t\}$ un trabajo J_j con tiempo de procesamiento $t_j = z_j$, fecha de entresa $d_j = 0$ y peso $w_j = z_j$.

- Para cada $i \in \{1,\ldots,t-1\}$ un trabajo J'_i con $t'_i = 1$, $d'_i = i(b+1)$ y $w'_i = 2$.

  Habrá, por tanto $3t + (t-1) = 4t - 1$ trabajos.

Con esta construcción, vamos a demostrar que el problema de la 3-Partición tiene solución si y solo si existe una secuencia con valor $\sum w_i T_i \leq y$ con

$$y = \sum_{J \leq K} z_J z_K + \frac{1}{2}(t-1)\,tb$$

$$J, K = 1, \ldots, 3t$$

Si trabajamos sin los J'_i, al tener los J_i fechas de entresa nulas, la suma ponderada de tardanzas coincidirá con $\sum w_i C_i$, este criterio es minimizado por la resla WSPT, pero al ser los tiempos de procesamiento iguales a los

pesos todos los cocientes serán iguales a 1 y por tanto el orden de los trabajos no influirá sobre el criterio, por lo que

$$\sum_{i} w_i T_i = \sum_{j \leq k} z_j z_k \qquad j, k = 1, \ldots, 3t$$

Si insertamos J'_1 en una secuencia, los trabajos posteriores a él se completarán una unidad de tiempo después, por lo que en la suma aparecerán más sumandos formados con elementos z_j y como $\sum_{j \in S_i} z_j = b$ y $d_1 = (b + 1)$ se verificará que

$$\sum_{i} w_i T_i + w'_1 T'_1 = \sum_{j \leq k} z_j z_k + ((t-1)b + |L'_1|)$$

Insertando todos los J'_i se obtiene una secuencia con

$$\sum_{i} w_i T_i = y + \sum_{i=1}^{t-1} |L'_i|$$

por lo que

$$\sum_{i} w_i T_i \leq y$$

si y solo si los J'_i se completan en sus fechas de entresa ($i(b+1)$) y tal secuencia existe si y solo si los trabajos J_i pueden dividirse en t grupos cada uno de 3 trabajos con tiempo de completación del grupo igual a b, es decir, si y solo si el problema de la 3-Partición tiene solución.

[]

De aquí deducimos a fortiori que el problema genérico $n|1|\sum c_i$ es fuertemente NP-difícil y por tanto no existirán ni siquiera algoritmos seudopolinomiales para el mismo, salvo que P=NP.

El hecho de que el criterio tensa estructura aditiva ha provocado que diversos autores hayan abordado este problema valiéndose de la Programación Dinámica, podemos considerar

como primeros trabajos en esta línea los de Held & Karp (1962) y Lawler (1964).

Dado un subconjunto T del conjunto J de índices de los n trabajos, sea

$$C(T) = \sum_{i \in T} t_i$$

y

c(K,t) = coste en que incurre la tarea K si $C_K = t$.

Si f(T) es el costo de la secuencia de mínimo coste formada por los elementos de T , al aplicar el principio de la Programación Dinámica obtenemos la siguiente relación de recurrencia:

$$f(T) = \min_{K \in T} \{ f(T-\{K\}) + c(K,C(t)) \}$$

con $f(\emptyset) = 0$

que será válida para las secuencias semiactivas bajo las condiciones del modelo básico.

La anterior relación es del tipo hacia adelante ("forward") , análogamente puede razonarse hacia atrás ("backward").

Si definimos S_T como el tiempo más temprano en que los trabajos de T pueden comenzarse, supuesto que todos los que no están en T se procesan antes, tendremos

$$S_T = \sum_{i \notin T} t_i$$

y la relación de recurrencia toma la forma

$$F(T) = \min_{K \in T} \{F(T-\{K\}) + c(K, S_T + t_K) \}$$

con $F(\emptyset) = 0$

Para el caso de existir un único recurso de disponibilidad variable podemos generalizar las relaciones de recurrencia, así, para el caso de trabajar hacia adelante basta de-

finir c(T) como el tiempo correspondiente a un nivel de recursos

$$\sum_{i \in T} q_i \quad ,$$

que puede ser calculado por la función tiempo desarrollada en el apartado anterior.

El elemento básico para la implementación de estas ecuaciones es la forma de representar los conjuntos . Baker (1974) asocia a cada conjunto T un número entero tal que su representación binaria nos da los elementos de T . Así, por ejemplo, para n=3 tenemos la siguiente equivalencia entre números y subconjuntos:

Entero	Binario	Subconjunto
0	000	{ }
1	001	{1}
2	010	{2}
3	011	{2,1}
4	100	{3}
5	101	{3,1}
6	110	{3,2}
7	111	{3,2,1}

Este sistema puede interpretarse como el de asignación de una etiqueta L[i] para cada trabajo J_i de forma que la etiqueta (entero correspondiente) es la suma de las etiquetas de los trabajos que lo constituyen, siendo

$$L[i] = 2^{i-1} \qquad \text{con} \quad i=1,\ldots,n \quad .$$

Dado un entero z , el procedimiento conjunto obtiene el subconjunto correspondiente, de forma que en la variable tam almacena el cardinal del subconjunto y en el vector com[K] (K=1,...,tam) los índices de los trabajos contenidos en él.

```
procedure conjunto(z:integer;var tam:integer;
                   com:array[1..n] of integer);
var K,i,ir:integer;
begin
  i:=z;
  tam:=0;
  K:=1;
  while i<>0 do
     begin
        ir:=i-2*(i div 2);
        i:=i div 2;
        if ir = 1 then
```

```
                                begin
                                  tam:=tam+1;
                                  com[tam]:=k
                                end;
                K:=K+1
            end
        end;
```

Utilizando esta herramienta hemos desarrollado un procedimiento que razonando hacia atrás almacena en un vector p la secuencia óptima para el problema del costo total.

El número de subconjuntos para un problema de tamaño n será 2^n, como habrá $2^n - 1$ de ellos no vacíos tomaremos esta constante (que denotaremos dim) como dimensión de los vectores F y pre que corresponden a los valores óptimos para cada subconjunto y a los índices de los trabajos anteriores a cada subconjunto respectivamente.

```
procedure  PD;
var  p,com,L : array[1..n] of integer;
     F,pre : array[1..dim] of integer;
     z,Sq,Q,T,i,tam,crit : integer;
begin
 Sq:=0;       {inicialización}
 L[1]:=1;
 for i:=2 to n do begin
                    L[i]:=2*L[i-1];
                    Sq:=Sq+q[i]
                  end;
 T:=tiempo(Sq);
 F[1]:=c(1,T);
 for z:=1 to dim do    {cálculo de  F}
    begin
     conjunto(z,tam ,com);
     Q:=Sq;
     for i:=1 to tam do Q:=Q-q[com[i]];
     F[z]:=maxint;
     for i:=1 to tam do
        begin
           T:=tiempo(Q+q[com[i]]);
           crit:=c(i,T);
           zi:=z-L[i]; {etiqueta del subconjunto obtenido
                        al quitar el elemento i}
           if zi<>0 then crit:=crit+F[zi];
           if crit<F[z] then begin
                                pre[z]:=com[i];
                                F[z]:=crit
                             end
        end
     end;
 p[1]:=pre[dim];
 for i:=2 to n do   {construcción de la secuencia óptima}
              begin
                z:=z-L[p[i-1]];
                p[i]:=prec[z]
              end
end;
```

Para el caso particular de ser el criterio la minimización de la tardanza ponderada media, la función c(i,T) toma la siguiente forma:

```
function  c(i,T:integer):integer;
begin
  c:=0;
  if T>d[i] then c:=w[i]*(T-d[i])
end;
```

y si es el de la minimización del número ponderado de trabajos tardíos la condición que aparece en la función será:

```
if T>d[i] then c:=w[i]
```

El procedimiento es del tipo $O(n2^n)$, con una necesidad de memoria $O(2^n)$. Esto hace que el método sea inviable para $n > 14$ en un ordenador con 64 Kb y duplicando la memoria solo se logrará aumentar en una unidad el tamaño máximo de los problemas que pueden resolverse por este método.

Como hemos visto, la existencia de relaciones de precedencia aumenta la dificultad computacional de los problemas del tipo $n\_1\_\overline{\overline{\geq}}\ c_i$, sin embargo veremos cómo estas relaciones pueden ser explotadas para aumentar considerablemente la eficacia del método basado en la Programación Dinámica. Aún más, para un problema sin relaciones de precedencia podemos obtener ciertas relaciones de orden entre sus trabajos como consecuencia de la verificación de determinadas "condiciones de dominancia" del tipo:

'' si se verifica la condición C entonces J_i precede a J_j en una secuencia óptima '' .

El pionero en el desarrollo de este tipo de condiciones es Emmons (1969), que estudia el problema de la tardanza total. Posteriormente Rinnooy Kan, Lageweg y Lenstra (1975) generalizan las condiciones al caso de una función de coste arbitraria.

A continuación se estudian algunas condiciones de dominancia para el caso $n\_1\_rec\ var\_\overline{\overline{\geq}}\ c_i$.

Denotamos:

A_i : conjunto de índices de los trabajos que son posteriores al J_i en una secuencia óptima.

B_i : conjunto de índices de los trabajos para los que se ha demostrado que preceden a J_i en una secuencia óptima.

Para cualquier subconjunto S , $S' = \{1 ,..., n\} - S$.

$C(S)$: tiempo de completación de los trabajos en S, vendrá dado por

$$\text{tiempo} \left(\sum_{J \in S} q_j \right)$$

Por acortar las expresiones escribimos $t(z) = \text{tiempo}(z)$.

Teorema ------- Si para dos trabajos J_j y J_K se verifica:

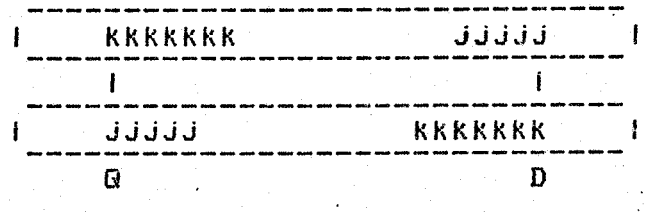
1) $c_J(t) - c_K(t)$ es una función no decreciente en el intervalo $(C(B_K \cup \{K\}) , C(A'_J))$, y

2) $q_j \leq q_K$

Entonces J_j precede a J_K en una secuencia óptima.

Demostración ------------

Suponsamos que en una secuencia J_K precede a J_j , vamos a intercambiar ambos trabajos como aparece en la siguiente figura:



donde en la secuencia inicial tenemos que Q es el nivel de recursos antes de empezar a procesarse el trabajo K y D es el nivel de recursos después de completar el trabajo J.

Se verificará que

$$\sum_{j \in B_K} q_j + q_K \leq Q + q_K \leq D \leq \sum_{i \in A'_j} q_i$$

y por ser la función tiempo no decreciente

$$C(B_K \cup \{K\}) \leq t(Q + q_K) \leq t(D) \leq C(A'_j)$$

por la condición 1)

$$c_j(t(D)) - c_K(t(D)) \geq c_j(t(Q+q_K)) - c_K(t(Q+q_K))$$

por la condición 2)

$$c_j(t(Q+q_K)) \geq c_j(t(Q+q_j))$$

y por tanto

$$c_j(t(D)) + c_K(t(Q+q_K)) \geq c_j(t(Q+q_j)) + c_K(t(D))$$

luego si intercambiamos los trabajos, el costo total decrece o permanece igual.

[]

Basta aplicar el teorema al caso de ser los conjuntos B_K y A_i vacíos para obtener el siguiente coralario para el criterio de minimización de la tardanza ponderada media.

Corolario ---------- Si $c_i(t) = w_i \max\{0, t - d_i\}$ y

$$d_j \leq d_K \quad , \quad w_j \geq w_K \quad ; \quad t_j \leq t_K$$

Entonces J_j precede a J_K en una secuencia óptima.

Teorema ------- Si para dos trabajos J_j y J_K se verifica:

a) $c_K(C(B_K \cup \{K\})) = c_K(C(A'_j - \{K\}))$, y

b) $c_i(t) - c_j(t)$ es una función no decreciente en el intervalo $(C(A'_j - \{K\}), C(A'_j))$.

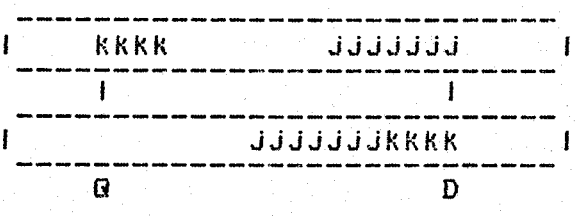
Entonces J_j precede a J_K en una secuencia óptima.

Demostración — Las condiciones a) y b) implican la condición 1) del teorema anterior.

Si $q_j \leq q_K$ podemos aplicar el citado teorema.

Si $q_j > q_K$, suponemos que tenemos una secuencia en la que el trabajo K precede al J.

Consideremos el intercambio que aparece en la figura



la contribución al coste total de todos los trabajos, excepto posiblemente el K, no aumenta.

Tenemos que

$$\sum_{i \in B_K} q_i + q_K \leq Q + q_K \leq D - q_K \leq \sum_{i \in A'_j} q_i - q_K$$

lo que junto con la condición a) nos dá

$$c_K(t(D-q_K)) = c_K(t(Q+q_K))$$

y por b)

$$c_j(t(D)) + c_K(t(Q+q_K)) > c_j(t(D-q_K)) + c_K(t(D))$$

por lo que el costo total decrece o permanece igual.

□

Consecuencia inmediata es el siguiente

Corolario — Si para los trabajos J y K se verifica:

$$c_K(C(B_K \cup \{K\})) = c_K(C(A'_j))$$

entonces el trabajo J precede al K en una secuencia óptima.

En particular si $c_K(t(q_K)) = c_K(t(S_q))$ el trabajo J $_K$

es el último en una secuencia óptima.

A partir de estos teoremas y corolarios se obtienen como consecuencia inmediata los teoremas dados por Baker (1980) para el caso de la tardanza ponderada con un único recurso variable, resultados que a su vez son una generalización de las condiciones de dominancia de Emmons.

Cuando se aplican reiteradamente las condiciones de dominancia, pueden provocarse ciclos. Para evitar ésto, una forma es considerar, en cada etapa, solamente aquellos pares de trabajos que no hayan sido relacionados hasta ese momento, es decir, aquellos pares (i,j) tales que $i \notin B_j \cup A_j$.

Las condiciones de dominancia se utilizan de dos formas:

- Generando relaciones de precedencia que se aprovechan al aplicar la Programación Dinámica.
- Empleándolas como criterios de eliminación cuando se aplican técnicas de ramificación y acotación.

La primera línea es la seguida por Srinivasan (1972) y Baker y Schrage (1978) , mientras que la segunda es la debida a Rinnoy Kan,Lageweg y Lenstra (1975).

De las experiencias computacionales realizadas por Schrage y Baker (1978) se deduce la mayor efectividad del método basado en la Programación Dinámica . Es este camino el que hemos seguido con el fin de disminuir las necesidades de memoria del método.

La primera cuestión a dilucidar es la de cuál es el sistema más eficiente de implementar las relaciones de precedencia (o lo que es lo mismo, de implementar un digrafo acíclico).

De entre los diversos métodos,los que dan mejor resultado son los dos siguientes:

- Matriz de adyacencia , $P = (p_{ij})$

  donde $p_{ij} = 1$ si $J_i << J_j$,

  $p_{ij} = 0$ en caso contrario.

- Lista de trabajos adyacentes,

  $A[i] = \{ J \mid J_i << J_j \}$.

El primer método requiere $O(n^2)$ posiciones de memoria .

El segundo necesitará una cantidad de memoria igual a la suma de las longitudes de las listas. Si el número de arcos del digrafo asociado es n_a , la cantidad total de memoria será $O(n_a \log n)$.

Como $n_a < \frac{n\,(n-1)}{2}$, dependerá de los valores de n_a el que un sistema u otro sea más eficaz.

En términos generales, podemos decir que es preferible utilizar las listas de elementos adyacentes cuando el digrafo es poco denso y la matriz de adyacencia cuando es muy denso.

Como medida de la densidad empleamos el siguiente índice:

$$d(P) = \frac{2\,n_a}{n\,(n-1)} * 100$$

donde

$$n_a = \sum_{i,j=1}^{n} p_{ij}$$

A continuación, damos un método de construcción de una matriz de adyacencia para el problema:

$$n \mid 1 \mid \text{reci var} \mid \sum w_i T_i \quad .$$

Suponemos ordenados los trabajos según la regla SPT, resolviendo los empates por la EDD.

Definimos las variables globales:

```
i,J,m : integer;
cont : boolean;
```

el procedimiento continuar nos dá la condición de terminación del algoritmo.

```
procedure continuar;
begin
   cont:=true;
   if m<J-2 then begin
                   i:=i+1;
                   if i=J then i:=1
                 end
            else begin
                   J:=J-1;
                   i:=1;
                   if J=1 then cont:=false
                 end
end;
```

Con el podemos escribir la siguiente función que nos dá la matriz de adyacencia:

```
function P : array[1..n,1..n] of 0..1;
var  R,Q : array[1..n] of integer;
begin          {inicializar}
  for i:=1 to n do for J:=1 to n do P[i,J]:=0;
  for i:=1 to n do begin
                     R[i]:=q[i];
                     q:=q+q[i]
                   end;
  for i:=1 to n do q[i]:=q;
  i:=1 ; J:=n;
  repeat       {construcción de la matriz}
   if P[i,J]=1 then  continuar;
               else
        if( (d[i]<=d[J]) or (d[i]<=tiempo(R[J])) )
          and (w[i]>=w[J])
            then begin
                   m:=0;
                   P[i,J]:=1;
                   R[J]:=R[J]+q[i];
                   Q[i]:=Q[i]-q[J];
                   i:=i+1;
                   if i=J then i:=1
                 end
          else
            if (d[i]>tiempo(Q[J]-q[i]) and (w[i]<=w[J])
               and (P[J,i]<>1)
                 then
                   begin
                     m:=0;
                     P[J,i]:=1;
                     R[i]:=R[i]+q[J];
                     Q[J]:=Q[J]-q[i];
                     i:=i+1;
                     if i=J then i:=1
                   end
                 else continuar;
  until cont
end;
```

Una vez ordenados los trabajos como se ha dicho, el procedimiento opera testando las condiciones de dominancia siguientes:

- Si $w_i \geq w_j$ y $d_i \leq \max (d_j , C(B_i \cup \{J\}))$

  entonces J_i precede a J_j en una secuencia óptima.

- Si $w_j \leq w_i$ y $d_j \geq \max (d_i , C(A'_i - \{J\}))$

  entonces J_j precede a J_i en una secuencia óptima.

Ambos son consecuencias de nuestros teoremas para el caso de ser el criterio la tardanza total ponderada, es decir:

$$c_i(t) = w_i \max(0, t-d_i) \quad .$$

Es preferible, sin embarso, de cara a la implementación de los algoritmos trabajar , en vez de con la matriz de adyacencia, con la matriz de conexión , que denotaremos por M y que viene dada por

$$m_{ij} = 1 \quad \text{si} \quad J_i < J_j$$

$$m_{ij} = 0 \quad \text{en otro caso.}$$

La obtención de M a partir de P , la realiza la función $\underset{\sim}{M}$, la cual utiliza las funciones auxiliares $\underset{\sim}{m}$, $\underset{\sim\sim}{PR}$ y $\underset{\sim}{s}$ que nos dan, respectivamente, el máximo entre dos números, el "producto" y la "suma" de dos matrices (con las operaciones producto ordinario y el máximo como operación suma).

```
function  m(i,J:0..1) : 0..1;
begin
   if i>J then m:=i
          else m:=J
end;
```

```
function pr(a,b:array[1..n,1..n] of 0..1) :
          array[1..n,1..n] of 0..1;
var i,J,K : inteser;
besin
  for i:=1 to n do
  for J:=1 to n do
   besin
     pr[i,J]:=0;
     K:=1;
     while (K<=n) and (pr[i,J]=0) do
        besin
          pr[i,J]:=m(pr[i,J],a[i,K]*b[K,J]);
          K:=K+1
        end
   end;
end;


function s(a,b:array[1..n,1..n] of 0..1) :
          array[1..n,1..n] of 0..1;
var i,J : inteser;
besin
   for i:=1 to n do
   for J:=1 to n do
      s[i,J]:=m(a[i,J],b[i,J])
end;


function  M(P:array[1..n,1..n] of 0..1) :
          array[1..n,1..n] of 0..1;
var Pr:array[1..n] of 0..1;
besin
   Pr:=P;
   M:=P;
   for i:= 2 to n do
              besin
                 Pr:=pr(Pr,P);
                 M:=s(M,Pr)
              end
end;
```

Con ésto, podemos comenzar el desarrollo del algoritmo de Programación Dinámica para el caso de existir relaciones de precedencia.

El concepto fundamental sobre el que se basa nuestro algoritmo es el de conjunto factible de trabajos. Decimos que un conjunto de trabajos S c J , es factible si, para todo trabajo K de S, todos los predecesores de K están también en S.

Denotamos por T(S) al conjunto de trabajos de S sin sucesores en S (trabajos terminales).

La recursión de la Programación Dinámica toma la siguiente forma:

$$f(S) = \min_{K \in T(S)} \{ f(S-\{K\}) + s(K,C(S)) \}$$

donde S es un subconjunto factible.

Para implementar el algoritmo necesitamos:

- Un esquema de numeración de los trabajos.
- Un procedimiento de generación de todos lo subconjuntos factibles de forma que S-{K} se genere antes de S.
- Un procedimiento de direccionamiento tal que, dado S, f(S) pueda ser obtenido rápidamente.

Estos tres componentes dependerán fundamentalmente, de la estructura que utilicemos para almacenar los valores f(S). Podemos considerar dos casos:

I) Que utilicemos un fichero secuencial indexado.

Desde el punto de vista operativo,este es el mejor sistema por cuanto evita al analista el tener que desarrollar los procedimientos de búsqueda.

En este caso, numeramos los trabajos de forma que si $c_{ij} = 1$ entonces N[i] < N[J] (donde N[k] es el número asociado por el esquema de numeración al trabajo K) . Esto siempre será posible por cuanto que un digrafo es acíclico si y solo si admite un tal esquema de numeración.

El sistema de etiquetado consiste, al igual que en el caso de no existir relaciones de precedencia, en la asignación de un número entero (etiqueta) a cada trabajo, de forma que la etiqueta correspondiente a un conjunto sea la suma de las etiquetas de los trabajos componentes . El procedimiento de etiquetado lo realiza el procedimiento etiq .

Definimos :

b[K] = suma de las etiquetas de los trabajos i / $m_{iK} = 1$.

a[K] = suma de las etiquetas de los trabajos i / $m_{Ki} = 1$.

T[K] = suma de las etiquetas de los trabajos i / i < K .

L[K] = etiqueta del trabajo K .

como vectores de n componentes enteras de caracter global, salvo el T que tiene n+1, siendo T[n+1] la etiqueta del conjunto de los n trabajos; el procedimiento será:

```
procedure etiq(M:array[1..n,1..n] of 0..1);
var J,K : integer;
begin
   for J:=1 to n do
      begin      {inicializar}
         T[J]:=0;
         a[J]:=0;
         b[J]:=0
      end;
   for J:=1 to n do
      begin      {etiquetado}
         L[J]:=T[J]-a[J]-b[J]+1;
         for K:=J to n do
            begin
               if M[J,K]=1 then b[K]:=b[K]+L[J];
               if M[K,J]=1 then a[K]:=a[K]+L[J]
            end;
         T[J+1]:=T[J]+L[J]
      end
end;
```

Este procedimiento hace corresponder a cada etiqueta un único subconjunto, para demostrarlo basta razonar por inducción sobre n.

El método de generación de subconjuntos factibles vuelve a ser una generalización del caso de no existir relaciones de precedencia y lo realiza el procedimiento miem.

Definimos las variables globales:

m : array[1..n+1] of integer (para i = 1 ,..., n m[i] = 1 si el trabajo i está en el conjunto y 0 en otro caso).

cont : boolean (si cont = false se han generado todos los subconjuntos factibles).

```
procedure miem(var m : array[1..n] of integer);
var i,J,K,s : integer;
begin
   J:=1 ; m[n+1]:=0;
      while  m[J]<>0 do J:=J+1;
      if J<=n then
               begin
                  i:=J;
                  m[i]:=1;
                  for J:=i-1 downto 1 do
                        begin
                           s:=0;
                           for K:=1 to n do s:=s+M[J,K]*m[K];
                           if s=0 then m[J]:=0
                        end
               end
              else cont:=false
end;
```

Las versiones estandar de la mayor parte de los lenguajes científicos de programación carecen de la estructura de fichero indexado, de la cual disponen lenguajes de gestión tales como el COBOL.

Al no disponer de versiones de PASCAL con esta estructura hemos implementado el algoritmo de Programación Dinámica en el FORTRAN de un VAX 11, el listado del programa aparece como anexo.

Las necesidades de memoria vendrán determinada por el número de subconjuntos factibles n_s. La función de complejidad del tiempo será $O(n\ n_s)$.

II) Que utilicemos un vector.

Este es el sistema seguido por Baker y Schrage. Con este sistema se logran tiempos de búsqueda menores al utilizar memoria central. La principal dificultad de la implementación realizada por estos autores, es que la cantidad de memoria necesaria viene dada por el número de etiquetas n_e (que coincidirá con la suma de las etiquetas de los n trabajos).

Si el número de etiquetas coincide con el número de subconjuntos factibles, se dice que el etiquetado es compacto, siendo la situación óptima. Sin embargo, en general, no será esta la situación, por lo que la mejora lograda en el tiempo se ve contrarestada con un aumento de las necesidades de memoria igual a $n_e - n_s$.

Se intenta disminuir el número de etiquetas modificando el esquema de numeración, asignando en cada etapa el siguiente número al trabajo tal, que al ser etiquetado, reciba la menor etiqueta posible. En esta línea Burns y Steiner (1981) dan una modificación del esquema de numeración para el cual

demuestran que obtiene etiquetados compactos para grafos serie-paralelos.

Nuestra alternativa para el caso de una red general consiste en utilizar un vector cuya dimensión sea igual al número de subconjuntos factibles, y cuyos elementos sean registros en los que además del valor f(S) y el índice del trabajo predecesor en la secuencia óptima, esté contenida la etiqueta correspondiente.

```
reg = record
        f,pre,et:integer
      end;

F : array[1..ns] of reg;
```

Será preciso desarrollar un método que transforme etiquetas en índices del vector. Consideramos que los sistemas más eficientes son la búsqueda binaria y el empleo de tablas "hash" .

La búsqueda binaria ya se empleó como base de la función tiempo y como dijimos, requiere para cada búsqueda $\log_2 n$ operaciones como máximo.

El método de la tabla "hash" consiste en construir una función hash H, tal que dada una etiqueta,obtenga el índice asociado. Una solución es utilizar la función

$$H(e) = e \bmod n_s$$

que tiene la propiedad de distribuir uniformente las etiquetas en el campo de índices. Evidentemente, pueden darse dos situaciones:

- F[H(e)].et = e , en cuyo caso hemos encontrado el índice correspondiente.
- F[H(e)].et ≠ e , decimos entonces que se produce una "colisión".

Sistemas clásicos para el manejo de colisiones son:

- Inspección lineal:

$$h_0 = H(K)$$

$$h_i = (h_0 + i) \bmod n_e \quad , i=1,\ldots,n_e - 1$$

- Inspección cuadrática:

$$h_0 = H(K)$$

$$h_i = (h_0 + i^2) \bmod n_e \quad , \quad i > 0$$

El análisis de estos métodos puede encontrarse en Wirth (1976) y Knuth (1973).

Para aprovechar la estructura del conjunto de etiquetas, hemos desarrollado el siguiente método:

```
if K > n_s  then  begin
                     h_0 = H(K);
                     h_i = (h_0 + i) mod n_e
                  end
             else begin
                     h_0 = H(K);
                     h_i = h_0 - i
                  end
```

para $i = 1,\ldots,n_e$.

Dada una etiqueta, el cálculo del índice asociado, utilizando este método, lo realiza la función h.

El procedimiento PD_prec desarrolla el método de Programación Dinámica para el caso de relaciones de precedencia con un único recurso de disponibilidad variable ,el cual supone que es conocida la matriz de conexión M.

```
procedure PD_prec;
var p,L,m:array[1,.n] of integer;
    i,J,ii,z,iz,zi,nsuc,s,Q:integer;
    cont:boolean;
begin
   cont:=true;
   etiq(M);
```

```
    for i:=1 to n do m[i]:=0;
    with F[1] do
     begin
       s:=c(1,tiempo(q[1]));
       pre:=1;
       et:=1
     end;
    iv:=0;
    while cont do
       begin
         iv:=iv+1;
         z:=0;
         miem(m);
         for i:=1 to n do z:=z+m[i]*L[i];
         F[iv].et:=z;
         F[iv].f:=maxint;
         Q:=0;
         for i:=1 to n do Q:=Q+m[i]*q[i];
         for i:=1 to n do
            begin
             iz:=h(z);
             if m[i]=1 then
               begin
                  nsuc:=0;
                  for ii:=i to n do nsuc:=nsuc+M[i,ii]*m[ii];
                  if nsuc=0 then
                    begin
                       s:=c(i,tiempo(Q));
                       z1:=z-L[i];
                       if z1<>0 then
                         begin
                            s:=s+F[h(z1)].f;
                            if s<F[iz].f then
                                  begin
                                     F[iz].f:=s;
                                     F[iz].pre:=i
                                  end
                         end
                    end
               end
            end
       end;
       p[n]:=F[h(z)].pre
       for i:=n-1 downto 1 do
            begin
               z:=z-L[p[i+1]];
               p[i]:=F[h[z]].pre
            end
end;
```

De todo lo anterior se deduce, que la efectividad del método basado en la Programación Dinámica dependerá fundamentalmente del número de subconjuntos factibles o del número de etiquetas, según sea el enfoque.

Hemos estudiado la relación existente entre densidad del grafo asociado con las relaciones de precedencia y los valores de n_s y n_e. Para lo que se recurrió a un modelo de regresión del tipo

$$n_s = \frac{a}{d} + b \quad , \qquad n_e = \frac{a'}{d} + b' \quad .$$

Debido a que el valor n_e puede ser superior a maxint, trabajamos, en vez de con las cantidades n_e y n_s, con:

$$\frac{n_e}{2^n} * 100 \quad y \quad \frac{n_s}{2^n} * 100$$

que representan los porcentajes respecto al total de etiquetas para el caso de no existir relaciones de precedencia. Para la estimación de los parámetros a,a',b,b' se emplearon dos conjuntos de 30 problemas test de tamaño 15 y 25, cuyas matrices de advacencia se generaron aleatoriamente utilizando el procedimiento SGMA, el cual utiliza el procedimiento auxiliar:

```
procedure nal(var i,na:real);
var j:integer;
begin
   j:=trunc(i*s1);
   if j>=0 then j:=j+maxint;
   na:=j/maxint;
   i:=j
end;
```

donde s1 es una constante dependiente del número de bits que tenga la palabra en el ordenador que empleemos. Para el caso de tener 4 Bytes, un posible valor de s1 es 5^{13}.

```
procedure SGMA(var P:array[1..n,1..n] of 0..1);
var i,j:integer;
    prob,a,au:real;
begin
   read(au);
   for i:=1 to n do for j:=1 to n do P[i,j]:=0;
   nal(au,prob);
   for i:=1 to n do
   for j:=i+1 to n do
      begin
         nal(au,a);
         if a<prob then P[i,j]:=1
      end
end;
```

Los resultados obtenidos con los problemas tests generados de esta forma son:

| n = 15 |

a = 139.13
b = -4.09

r=0.98

a'= 154.66
b'= 4.25

r=0.85

n = 25

a = 3.17
b = -0.10

r=0.77

a'= 175.75
b'= -4.36

r=0.85

Lo que nos indica que el número de etiquetas es mucho mayor que el número de subconjuntos factibles a medida que crece el tamaño del problema.

APENDICE

Hemos definido formalmente algoritmo como un programa que es capaz de detenerse bajo cualquier configuración del problema.

Por tanto, un algoritmo para un problema de decisión puede considerarse como una correspondencia entre el conjunto de todas las posibles configuraciones del problema y el conjunto {sí,no} . Y un algoritmo para un problema general ("de búsqueda") será una correspondencia que asigna a cada configuración una solución.

A fin de expresar los algoritmos en un lenguaje que a la vez que fácilmente comprensible sea lo más cercano posible al utilizado en su implementación en el ordenador y que nos permita recurrir a la metodología dada por la Programación Estructurada, podemos elegir entre diversos sistemas. Entre los más extendidos están los "seudolenguajes" basados en el ALGOL, tales como el <<pidgin algol>>, término introducido por Aho,Hopcroft & Ullman (1974), o el <<cuasialgol>> empleado por Rinnooy Kan (1976); o bien utilizar un lenguaje de alto nivel tal como ALGOL, PL/I, ADA o PASCAL.

En la presente memoria nos hemos decidido por el empleo del PASCAL[*], debido a que tratamos de expresar la mayor parte de los algoritmos estudiados de forma que sean directamente ejecutables en una amplia gama de ordenadores, por lo que se precisa un lenguaje de alto nivel, y basamos nuestra elección entre ellos en el hecho de ser el PASCAL el que presenta una mayor variedad y potencia en las estructuras de datos disponibles (arrays,registros,ficheros,conjuntos,punteros y sus derivados).

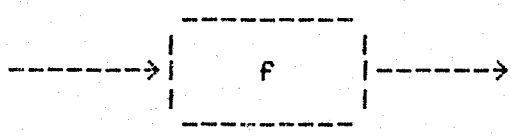
[*] El lenguaje PASCAL ha sido desarrollado por N. Wirth y las primeras versiones datan de 1971. Como texto de referencia hemos empleado:

K. Jensen & N. Wirth : PASCAL User Manual and Report. Lecture Notes in Computer Science n.18,1974.

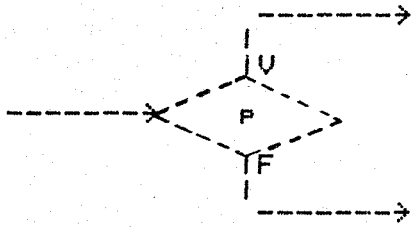
A continuación damos los elementos básicos de la Programación Estructurada, que con parecida o idéntica denominación son comunes a todos los lenguajes citados.

Un <<diagrama de flujo>> es una red dirigida con tres tipos de vértices:

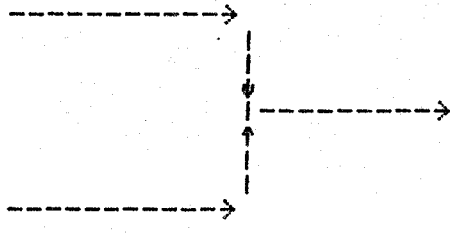
a) Vértice función, se utiliza para representar funciones f : X ---> Y , con X,Y conjuntos cualesquiera.



b) Vértice predicado, se utiliza para representar funciones booleanas p : X ---> { V, F } ,
p será un evaluador de expresiones booleanas y este tipo de vértice permite pasar el control a una u otra rama.



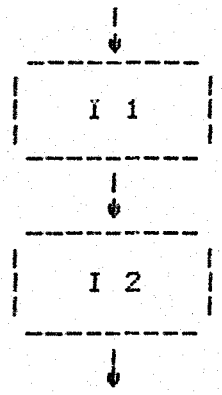
c) Vértice colector, que representa el paso del control desde dos ramas de entrada a una única de salida.



Un <<diagrama estructurado>> es aquel que puede ser expresado como combinación de los cuatro <<diagramas estructurados primitivos>> siguientes:

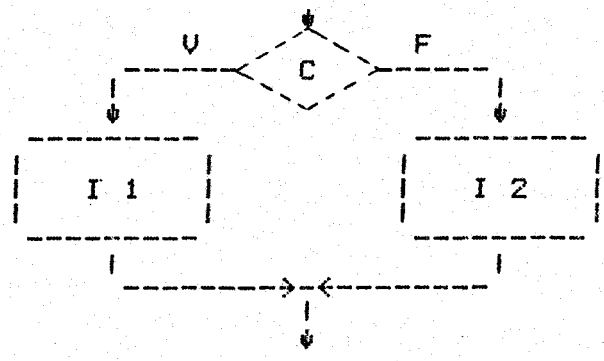
I) Composición

besin I1 ; I2 end



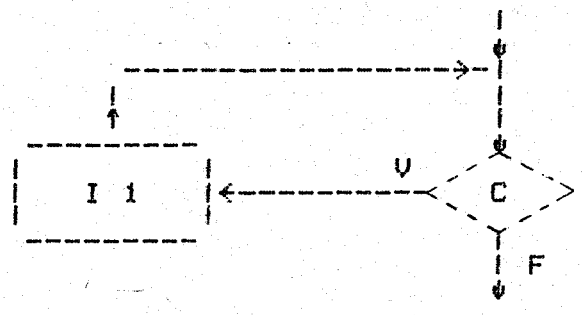
II) Selección

if C then I1
else I2



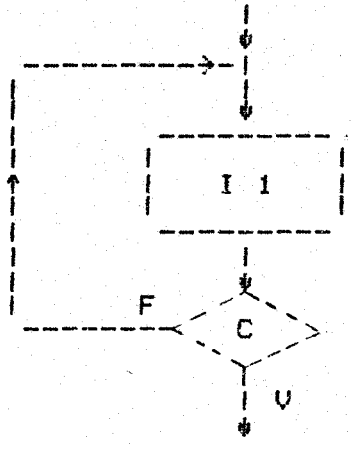
III) Iteración

while C do I1



IV) Iteración

repeat I1 until C



El teorema fundamental de la Programación Estructurada es debido a Bohm y Jacopini y afirma que cualquier diagrama de flujo puede ser representado utilizando únicamente los diagramas estructurados primitivos.

Para mayor comodidad en la escritura de programas se utiliza otra instrucción de iteración denominada "for". La instrucción for indica que una instrucción es repetidamente ejecutada mientras se asigna una progresión de valores a una variable denominada "variable de control". Tiene dos formas:

```
for v:=e1 to e2 do I
for v:=e1 downto e2 do I
```

en función de que e1 sea menor que e2 o se dé la situación inversa. La variable de control es v , los valores extremos que puede tomar son e1 y e2, y la instrucción que se ejecuta es la I.

El símbolo " := " es el de asignación, y los comentarios se escriben entre llaves ("{.....}") .

```
C     ------------------------------------------------------------------
C      PROGRAMA PARA LA OBTENCION DE LA SECUENCIA OPTIMA, UTILIZANDO
C      LA PROGRAMACION DINAMICA, PARA EL CRITERIO DEL COSTO TOTAL
C      CUANDO EXISTEN RELACIONES DE PRECEDENCIA Y UN UNICO RECURSO
C      VARIABLE
C     ------------------------------------------------------------------
C
C     LA SIGUIENTE INSTRUCCION DEPENDERA DE LA DIMENSION DEL PROBLEM
C
01          PARAMETER N=25
C
C     Y LA SIGUIENTE DE LA LONGITUD DE LA PALABRA DEL ORDENADOR
C
02          PARAMETER MAXINT=32766
C
03          IMPLICIT INTEGER (A-Z)
C
C     P  ES LA MATRIZ DE CONEXION
C
04          BYTE P(N,N),FICH(10),M(N)
05          INTEGER POPT(N),L(N)
06          COMMON Q(N),D(N),W(N)

07          OPEN(UNIT=1,ACCESS='KEYED',STATUS='NEW',INITIALSIZE=20,
            1ORGANIZATION='INDEXED',FORM='UNFORMATTED',KEY=(1:4:INTEGER),
            2RECL=12)
08          TYPE 1000
09    1000  FORMAT(' NOMBRE DEL FICHERO DE LECTURA?'$)
10          ACCEPT 1001,FICH
11    1001  FORMAT(10A1)
12          OPEN(UNIT=2,NAME=FICH,TYPE='OLD',ACCESS='SEQUENTIAL')
C
C     LECTURA DE DATOS
C
13          DO 2 I=1,N
14    2      READ(2,21) (P(I,J),J=1,N)
15    21    FORMAT(<N>I1)
16          DO 3 I=1,N
17    3      READ(2,22) Q(I),D(I),W(I)
18    22    FORMAT(3I6)
C
C     ETIQUETADO
C
19          CALL ETIQ(P,N,L)
C
C     PRIMERA ETAPA
C
20          I=1
21          T=TIEMPO(Q(I))
22          F=C(I,T)
23          WRITE(1) I,I,F
24          M(1)=1
25          DO 5 I=2,N
26    5      M(I)=0
C
C     OBTENCION DEL SIGUIENTE SUBCONJUNTO FACTIBLE Y DEL VALOR
C     DEL FUNCIONAL ASOCIADO
```

```
      C
27    100   CALL MIEM(P,N,M,FIN)
28          IF(FIN) 50,50,200
29    50    J=0
30          QI=0
31          DO 51 I=1,N
32           J=J+M(I)*L(I)
33    51     QI=QI+M(I)*Q(I)
34          F=MAXINT
35          DO 150 I=1,N
36           IF(M(I).EQ.0) GOTO 150
37           NSUC=0
38           DO 151 II=1,N
39    151     NSUC=NSUC+P(I,II)*M(II)
40           IF(NSUC.EQ.0) GOTO 150
41           T=TIEMPO(QI)
42           CRIT=C(I,T)
43           J1=J-L(I)
44           IF(J1.EQ.0) GOTO 130
45           READ(1,KEY=J1) J1,PR,G
46           CRIT=CRIT+G
47    130    IF(CRIT.GE.F) GOTO 150
48           F=CRIT
49           IP=I
50    150   CONTINUE
51          WRITE(1) J,IP,F

52          GOTO 100
      C
      C     CONSTRUCCION DE LA SECUENCIA OPTIMA
      C
53    200   READ(1,KEY=J) J,POPT(N),F
54          DO 2000 I=N-1,1,-1
55           J=J-L(POPT(I+1))
56    2000    READ(1,KEY=J) J,POPT(I),FI
57          STOP
58          END
```

BIBLIOGRAFIA

ADOLPHSON,D.L.(1977) Single machine job sequencing with precedence constraints SIAM J. Computing,6, 40-54.

ADOLPHSON,D.L. y HU,T.C. (1973) Optimal linear ordering. SIAM J. Appl. Math.,25,403-423.

AGGARWAL,S.C. (1982) A focussed review of scheduling in services. Eur. J. Opl. Res.,9,114-121.

AKERS,S.B. (1956) A graphical approach to production scheduling problems. Ops. Res.,4,244-245.

BAKER,K.R. (1974) Introduction to Sequencing and Scheduling. John Wiley. & Sons.

BAKER,K.R. (1975) A comparative survey of flow-shop algorithms. Ops. Res.,23,62-73.

BAKER,K.R. y NUTTLE,H.L.W. (1980) Sequencing independent jobs within a single resource. Nav. Res. Logist. Q.,27,499-510.

BAKER,K.R. y SCHRAGE,L.E. (1978) Finding an optimal sequence by dynamic programming:an extension to precedence-related tasks. Ops. Res.,26,111-120.

BAKSHI,M.S. y ARORA,S.R. (1969) The sequencing problem. Msmt. Sci.,16,B247-263

BALAS,E. (1969) Machine sequencing via disjuntive graphs: An implicit enumeration algorithm. Ops. Res.,17,941-957.

BANERJEE,B.P. (1965) Single facility sequencing with random execution times. Ops. Res.,13,358-364.

BANSAL,S.P. (1980) Single machine scheduling to minimise weighted sum of completion times with secondary criterion - a branch and bound approach. Eur. J. Opl. Res.,5,177-181.

BELLMAN,R. (1957) Dynamic Programming. Princeton University Press.

BRUNO,J.L. (1976) Sequencing jobs with stochastic task structures on a single machine. J. ACM,23,655-664.

BURNS,R.N. (1976) Scheduling to minimise the weighted sum of completion times with secondary criteria. Nav. Res. Logist. Q.,23,125-129.

BURNS,R.N. y STEINER,G. (1981) Single machine scheduling with series-parallel precedence constraints. Ops. Res.,29,1195-1207.

COFFMAN,E.G.Jr.,Ed.(1976) Computer and Job-shop Scheduling Theory. John Wiley. Nueva York.

CHO,Y. y SAHNI,S. (1981) Preemptive schedulins of independent jobs with release and due times on open,flows and job shops. Ops. Res.,29,511-522.

CONOLLY,B. (1981) Techniques in Operational Research. Ellis Horwood.

CONWAY,R.;MAXWELL,W.L. y MILLER,L.W. (1967) Theory of Schedulins. Addison-Wesley,Readins.Mass.

DAHL,O.J; DIJKSTRA,E.W. y HOARE,C.A.R. (1972) Structured Prosrammins. Academic Press. Nueva York.

DE,P. y MORTON,T.E. (1982) Schedulins to minimize maximun lateness on unequal paralell processors. Comput. & Ops. Res.,9,221-232.

EMMONS,H. (1975) A note on a schedulins problem with dual criteria. Nav. Res. Losist. Q.,22,615-616.

EMMONS,H. (1969) One machine sequencins to minimize certain functions of job tardiness. Ops. Res.,17,701-705.

EMMONS,H. (1975) One machine sequencins to minimize mean flow time with minimun number tardy. Nav. Res. Losist. Q.,22,585-592.

FISHER,M.L. (1976) A Dual alsorithm for the one-machine schedulins problem. Math. Prosr.,11,229-251.

FRENCH,S. (1982) Sequencins and Schedulins: An introduction to the mathematics of the Job-Shop. Ed. Ellis Horwood, Chichester.

GAREY,M.R. (1973) Optimal task sequencins with precedence constraints. Discr. Math.,4,37-56.

GAREY,M.R. y JOHNSON,D.S. (1979) Computers and intratability : A suide to the theory of NP-Completeness. Freeman.San Francisco.

GAREY,M.R. y JOHNSON,D.S. (1976) Schedulins tasks with nonuniform deadlines on two processors. J. ACM.,23,461-467.

GAREY,M.R. y JOHNSON,D.S. (1977) Two-processor schedulins with star-times and deadlines. SIAM J. Computins,6,416-426.

GAREY,M.R.;JOHNSON,D.S.;SIMONS,B.B. y TARJAN,R.E. (1981) Schedulins unit-time tasks with arbitrary release times and deadlines. SIAM J. Computins.,10, 256-269.

GAREY,M.R.;GRAHAM,R.L. y JOHNSON,D.S. (1978) Performance suarantees for schedulins alsorithms. Ops. Res.,26,3-21.

GELDERS,L.F. y KLEINDORFER,P.R. (1974) Coordinatins assresate and detailed schedulins in one-machine job-shop. Part I,Theory. Ops. Res.,22,46-60.

GELDERS,L.F. y KLEINDORFER,P.R. (1975) Coordinating aggregate and detailed scheduling in one-machine job-shop. Part II,Computation and Structure.Ops. Res.,23,312-324.

GLAZEBROOK,K.D. (1981) On nonpreentive strategies in stochastic scheduling. Nav. Res. Logist. Q.,28,289-300.

GLAZEBROOK,K.D. (1980) On single machine sequencing with order constraints. Nav. Res. Logist. Q.,27,123-130.

GLAZEBROOK,K.D. y NASH,P. (1976) On multiserver stochastic scheduling. J. Roy. Statist. Soc.,B38,67-72.

GOLDBERG,H.M. (1980) Jackson conjeture on Earliest Due Date scheduling. Math. Ops. Res.,5,460-466.

GONZALEZ,T. y JOHSON,D.S. (1980) A new algorithm for preemptive scheduling of trees. J. Assoc. Comput. Mach.,23,655-679.

GONZALEZ,T. y SAHNI,S. (1976) Open-shop scheduling to minimize finish time. J. Assoc. Comput. Mach.,23,655-679.

GONZALEZ,T. y SAHNI,S. (1978) Preemptive scheduling of uniform processor systems. J. Assoc. Comput. Mach.,25,92-101.

GORENSTEIN,S. (1972) An algorithm for project (job) sequencing with resource constraints. Ops. Res.,20,835-850.

GRAHAM,R.L.;LAWLER,E.L.;LENSTRA,J.K. y RINNOOY KAN,A.H.G. (1978) Optimization in deterministic sequencing and scheduling: a survey. En Interfaces between computer science and operations research. Ed. Mathematical Centre Tracts. Amsterdam.

GRAVES,S.C. (1981) A review of production scheduling. Ops. Res.,29,646-675.

GUPTA,J.N.D. (1976) Optimal flowshop schedules with no intermediate storage space. Nav. Res. Logist. Q.,23,235-243.

HELD,M. y KARP,R.M. (1962) A dynamic programming approach to sequencing problems. J. SIAM,10,196-210.

HEYDEN,L.V.D. (1981) Scheduling Jobs with exponential processing and arrival times on identical processor so as to minimize the expected makespan. Math. Ops. Res.,6,305-312.

HODGSON,T.J. y McDONALD,G.W. (1981) Interactive scheduling of a generalized flow-shop.Part III: Quantifying user objetives to create better schedules. Interfaces.--,35-41.

HORN,W.A. (1972) Single machine Job sequencing with treelike precedence orderings and linear delay penalties. SIAM J. Appl. Math.,23,189-202.

HURRION,R.D. (1978) An investigation of visual interactive simulation methods using the Job-shop scheduling problem. J. Opl. Res. Soc.,29,1085-1094.

IBARRA,F y KIM,T (1977) Heuristic algorithms for scheduling independent tasks on nonidentical processors. J. Assoc. Comput. Mach.,24,280-289.

ICHIMORI,T.;ISHII,H. y NISHIDA,T. (1981) Algorithm for one machine job sequencing with precedence constraints. J. Ops. Res. Soc. of Japan,24,159-168.

JACKSON,J.R. (1965) Scheduling a production line to minimize maximun tardiness Research Report, University of California at Los Angeles.

JAFFE,J.M. (1980) An analysis of preemptive multiprocessor job scheduling. Math. Ops. Res.,5,415-421.

JAFFE,J.M. (1980) Bounds on the scheduling of typed task systems. SIAM J. Comput.,9,541-551.

JOHNSON,S.M. (1954) Optimal two- and three-stage production schedules with set up times included. Nav. Res. Logist. Q.,1,61-68.

KADANE,J.B. y SIMON,H.A. (1977) Optimal strategies for a class of constrained sequential problems. Ann. Statist.,5,237-255.

KANET,J.J. (1981) Minimizing the average deviation of job completion times about a common due date. Nav. Res. Logist. Q.,28,643,651.

KARP,R.M. (1972) Reducibility among combinatorial problems. En Complexity of Computer Computations, Miller,R.E. and Thatcher,J.W. Eds. Plenum Press. Nueva York.

KISE,H.;IBARAKI,T. y MINE,H. (1978) A solvable case of the one-machine scheduling problem with ready and due times. Ops. Res.,26,121-126.

KNUTH,D.E. (1969) The Art of Computer Programming. Ed. Addison-Wesley,Reading. Mass.

KUNDE,M. (1981) Nonpreentive LP-scheduling on homogeneous multiprocessor systems. SIAM J. Comput.,10,151-172.

LABETOULLE,J. y LAWLER,E.L (1978) On preemptive scheduling of unrelated parallell processors by linear programming. J. ACM,25,612-619

LAND,A.H.;LAPORTE,G. y MILIOTIS,P. (1978) A unified formulation of the machine scheduling problem. Eur. J. Opl. Res.,2,32-35.

LAM,S. y SETHI,R. (1977) Worst case analysis of two scheduling algorithms. SIAM J. Comput.,6,518-536.

LAWLER,E.L. (1973) Optimal sequencing of a single machine subject to precedence constraints. Mgmt. Sci.,19,544-546.

LAWLER,E.L. (1976) Sequencing to minimize the wheisthed number of tardy jobs. RAIRO Inf.,10,27-33.

LAWLER,E.L.;LENSTRA,J.K. y RINNOOY KAN,A.H.G. (1981) Minimizing maximun lateness in a two-machine open-shop. Math. Ops. Res.,6,153-158.

LAWLER,E.L. y MOORE,J.M. (1969) A functional equation and its application to resource allocation and sequencing problems. Msmt. Sci.,16,77-84.

LENSTRA,J.K. y RINNOOY KAN,A.H.G. (1978) Complexity of scheduling under precedence constraints. Ops. Res.,26,22-35.

LENSTRA,J.K. y RINNOOY KAN,A.H.G. (1980) Complexity results for scheduling chains on a single machine. Eur. J. Opl. Res.,4,270-275.

LEUNG,J. (1982) On scheduling independent tasks with restricted execution times. Ops. Res.,30,163-171.

LOUVEAUX,F. (1982) Optimal scheduling of income tax prepayments under stochastic incomes. Eur. J. Opl. Res.,9,26-32.

MAXFIELD,M.W. (1981) Sequencing and scheduling in real time - Quickly. Interfaces,11,40-43.

MAXWELL,W.L. (1970) On sequencing n jobs on one machine to minimize the number of late jobs. Msmt. Sci.,16,295-297.

MAXWELL,W.L. (1972) A rejoinder. Msmt. Sci.,18,718-719.

McMAHON,G. y FLORIAN,M. (1975) On scheduling with ready times and due dates to minimize maximun lateness. Ops. Res.,23,475-482.

MIYAZAKI,S. (1981) One machine scheduling problem with dual criteria. J. Ops. Res. Soc. of Japan,24,37-50.

MIYAZKI,S y NISHIYAMA,N. (1980) Analisis for minimizing weisthed mean flowtime in flow-shop scheduling. J. Ops. Res. Soc. of Japan,23,118-132.

MONMA,C.L.(1982) Linear-time algorithms for scheduling on paralell processors. Ops. Res.,30,116-124.

MONMA,C.L. (1980) Sequencing to minimize the maximun job cost. Ops. Res.,28,942-951.

MOORE,J.M. (1968) An n-Job,one machine sequencing algorithm for minimising the number of late jobs. Msmt. Sci.,15,102-109.

PANWALKER,S.S. y ISKANDER,W. (1977) A survey of scheduling rules. Ops. Res.,25,45-61.

PANWALKER,S.S.;SMITH,M.L. y WOOLLAM,C.R. (1981) Counterexamples to optimal permutation schedules for certain flow-shop problems. Nav. Res. Logist. Q. 28,339-340.

PERL,Y. y YESHA,Y. (1981) Mean flow scheduling and optimal construction of a treelike communication network. Network,11,87-92.

PICARD,J.C. y QUEYRANE,M. (1981) On the one-dimensional space allocation problem. Ops. Res.,29,371-391.

PINEDO,M. (1981) Minimizing makespan with bimodal processing time distribution Msmt. Sci.,27,582-586.

PINEDO,M. (1982) Minimizing the expected makespan in stochastic flow shops. Ops. Res.,30,148-162.

PRISKERS,A.A.B.;WATTERS,L.J. y WOLFE,P.M. (1969) Multiproject scheduling with limited resources a cero-one programming approach. Msmt. Sci.,16,93-108.

PSFARATIS,H.N. (1981) A dynamic programming approach for sequencing groups of identical Jobs. Ops. Res.,29,1347-1359.

RINNOY KAN,A.H.G. (1976) Machine Scheduling Problems: Classification,Complexity and Computations. Martinus Nijhoff, The Hague.

RINNOY KAN,A.H.G.;LAGEWEG,B.J. y LENSTRA,J.K. (1975) Minimizing total costs in one-machine scheduling. Ops. Res.,23,908-927.

ROTHKOPF,M.H. (1966) Scheduling with random service times. Msmt. Sci.,12,707-713.

SALVADOR,M.S. (1978) Scheduling and sequencing. En Handbook of Operations Research: Models and Aplications, II,268-300.

SCHRAGE,L. y BAKER,K.R. (1978) Dynamic programming solution of sequencing problems with precedence constraints. Ops. Res.,26,444-449.

SEIDMANN,A y SMITH,M.L. (1981) Due date assignements for production systems. Msmt. Sci.,27,571-581.

SHAPIRO,R.D. (1980) Scheduling Coupled Tasks. Nav. Res. Logist. Q.,27,489-496.

SIDNEY,J.B. (1972) A comment on a paper of Maxwell. Msmt. Sci.,18,1972.

SIDNEY,J.B. (1975) Decomposition algorithm for single machine sequencing with precedence relations and deferral costs. Ops. Res.,23,283-298.

SIDNEY,J.B. (1981) A decomposition algorithm for sequencing with general precedence constraints. Math. Ops. Res.,6,190-204.

SMITH,W.E. (1956) Various optimizers for single state production. Nav. Res. Loist. Q.,3,59-66.

STURM,L.B.J.M. (1970) A single optimality proof of Moore's sequencing algorithm. Msmt. Sci.,17,B118-B118.

SWEENEY,D.J. y MURPHI,R.A. (1981) Branch & Baund methods for multi-item scheduling. Ops. Res.,29,853-864.

SZWARC,W. (1981) Extreme solutions of the two machine flow-shop problem. Nav. Res. Logist. Q.,28,103-114.

SZWARC,W. (1981) Precedence relations of the flow-shop problem. Ops. Res.,29,400-411.

VAN WASSENHOVE,L.N. y GELDERS,L.F. (1978) Four solution techniques for a general one-machine scheduling problem: a comparative study. Eur. J. Opl. Res. 2,281-290.

VAN WASSENHOVE,L.N. y GELDERS,L.F. (1980) Solving a bicriterion scheduling problem. Eur. J. Opl. Res.,4,42-48.

VICKSON,R.G. (1980) The single machine sequencing problems involving controlable job processing times. AIEE Trans.,12,258-262.

WEBER,R.R. y NASH,P. (1978) An optimal strategy in multi-server stochastic scheduling. J. Roy. Statist. Soc.,40,322-327.

WEEKS,W. (1979) A simulation study of predictable due-dates. Mgmt. Sci.,25,363-373.

WEISS,H.J. (1981) A greedy heuristic for single machine sequencing with precedence constraints. Mgmt. Sci.,27,1209-1216.

WIRTH,N. (1976) Algorithms + data structures = programs. Ed. Pretince-Hall.

UNIVERSIDAD DE [illegible]

FACULTAD DE CIENCIAS MATE[illegible]

Reunido el Tribunal integrado por los abajo firmantes el día de la fecha, para juzgar la Tesis Doctoral de D. José Luis Pino Mejías titulada "Problemas de [illegible] con fecha de enero", acordó otorgarle la calificación de Sobresaliente Cum Laude.

Sevilla, [illegible] de [illegible] 19[illegible]

El Vocal, El Vocal, El Vocal,

El Presidente, El Secretario, El Doctorado,