N°10
2EJ

UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

ESCUELA NACIONAL DE ESTUDIOS PROFESIONALES

" A C A T L A N "

APLICACION DE LA TECNICA DE RAMIFICACION Y ACOTACION A UN PROBLEMA DE ITINERACION

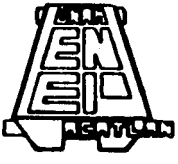
T E S I S

QUE PARA OBTENER EL TITULO DE

LIC. EN MATEMATICAS APLICADAS Y COMPUTACION

P R E S E N T A :

EDUARDO SCHMIDKONZ GARCIA



ESTADO DE MEXICO

FEBRERO DE 1992

UNAM – Dirección General de Bibliotecas Tesis Digitales Restricciones de uso

INDICE

INTRODUCCION.

" Así que la misión no es contemplar al mundo como jamás hombre alguno lo ha hecho, sino meditar como jamás hombre alguno lo ha hecho en la inmensidad del universo"

SCHOPENHAUER

En los años recientes, a nivel mundial, ha habido un notable incremento en el interés por los problemas de la itineración industrial. Es de suponerse que éstos han existido desde la creación de la primera fábrica, que estaba conformada por más de un trabajador o de una máquina. Hasta el año de 1954, no existía ningún intento de resolverlos analíticamente, desde entonces se han hecho modelos que representan problemas simples, formulados y analizados utilizando técnicas como análisis combinatorio, programación dinámica, programación entera, métodos heurísticos, análisis de redes, simulación y teoría de colas. La dificultad principal con estas aproximaciones ha sido el gran esfuerzo computacional que se incrementa rápidamente, aún con problemas pequeños y suposiciones muy simples. En la actualidad, debido al desarrollo de las técnicas computacionales, es posible resolver problemas de itineración cuya solución no existía o era ineficiente.

En un sistema de producción, la mayoría de los procesos de manufactura involucran varias operaciones, que realizan en alguna máquina la transformación de la materia prima en producto terminado, como por ejemplo el trazado de un molde de costura, su corte, y su confección. El problema de definir la transformación que va a ser realizada por cada una de estas operaciones es tecnológico, debido a que se puede contar con alguna máquina que realice 2 o 3 operaciones; al igual que el problema de establecer la secuencia de estas operaciones, i.e. es posible definir un conjunto de operaciones las cuales, desde un punto de vista tecnológico, pueden igualmente seguir una operación dada. En tal caso, es posible definir un subconjunto de soluciones tecnológicamente factibles que están de acuerdo con algún criterio económico, previamente establecido. Al problema de descubrir este tipo de secuencias se le denomina "problema de secuenciación", que formalmente consiste en determinar la secuencia en la cual un conjunto de trabajos va a ser ejecutado en una o más máquinas. Una secuencia no señala el tiempo en el cual los trabajos son ejecutados, ni tampoco la existencia de tiempos de ocio entre las operaciones. La secuenciación es sólo una parte del problema del control de la producción en un sistema, y se encuentra precedida por el área de planeación, que comprende funciones como:

procuración de material y de procesos tecnológicos, capacidad productiva y estimación de tiempos de proceso. Del mismo modo, es sucedida por el área de control que comprende la regularización y arreglo de las actividades de producción que conforman los planes. Debido a la fuerte interacción entre ella y sus áreas precedente y siguiente, no es obvio que se pueda aislar de cualquiera de las dos, y una decisión tiene una influencia relativa en cualquiera de ellas.

Cuando el problema de secuenciación contempla que la disponibilidad de la maquinaria varía en el tiempo debido a la ejecución de los trabajos, se le denomina: "problema de itineración". La resolución de éste, debe de establecer una serie de secuencias factibles de operaciones, las que serán realizadas en cada máquina a través del tiempo, especificando el tiempo de ocio (si existe), entre la ejecución de los trabajos. Las que en su totalidad conforman un itinerario que satisface un criterio económico. Existe confusión entre los términos secuenciación e itineración, sin embargo se pueden establecer las definiciones convenientes para señalar una distinción. La secuenciación está relacionada con los arreglos y permutaciones en donde un conjunto de trabajos bajo consideración, son ejecutados en todas las máquinas. La itineración está relacionada con la especificación de los tiempos de inicio y terminación de los trabajos en todas las máquinas. Las decisiones de secuenciación se encargan del arreglo de los eventos, mientras que las decisiones de itineración se encargan del tiempo de los eventos. Ambos términos, a pesar de ser diferentes, se usan frecuentemente como sinónimos, la razón es que en la mayoría de los casos se asume que cada trabajo es comenzado tan pronto como sea posible, entonces un itinerario es creado automáticamente, cuando una secuencia en particular es buscada.

En la actualidad, la industria mexicana tiene que hacer frente a grandes presiones comerciales, producidas por las nuevas políticas de Apertura Comercial y avances tecnológicos, lo que trae como consecuencia la necesidad de mejorar o sustituir los esquemas de producción. Una de las desventajas que padece el industrial mexicano, al igual que el pueblo en general, es el no saber planear con exactitud la realización de alguna actividad, por sencilla que sea, ni saber controlar el desarrollo o ejecución de las actividades conforme a los planes que hubieran podido ser determinados.

Esta situación, me condujó a desarrollar como proyecto de tesis un algoritmo de reciente aparición, realizado en el año de 1989, cuyos autores son los doctores Yang, Wang y Lee, el cual resuelve un problema de itineración que últimamente ha cobrado interés dentro del ámbito científico, el llamado "Problema de

itineración de actividades de igual tiempo de ejecución con restricciones de precedencia y perfil". El antecedente inmediato en la resolución de este problema es el "Algoritmo de programación dinámica de Dolev y Warmuth", elaborado en el año de 1982, el que al ser comparado con el nuevo, resalta su ineficiencia operativa.

En este algoritmo se utiliza la técnica de ramificación y acotación, la que se utiliza para resolver varios problemas de orígen combinatorio. Se escogió éste, porque presenta una nueva filosofía entre ese tipo de algoritmos, debido a que mientras todos los demás enfatizan el cálculo de la cota inferior (proceso de acotación), éste lo hace sobre la disminución del espacio solución a través de reglas de terminación, y por lo tanto la búsqueda de las soluciones factibles se reduce en gran medida. Además existe la posibilidad de utilizar sus reglas estratégicas para mejorar otros que presenten características similares.

La investigación de tesis se realiza a través de cuatro capítulos y dos apéndices. En el primer capítulo se presenta un panorama general de la teoría de itineración, para ello se trata de ubicarla dentro de los sistemas de producción, mostrando sus objetivos, los modelos de itineración y sus supuestos, concluyendo con los criterios de optimización más comunmente utilizados. El segundo capítulo, pretende explicar la técnica de ramificación y acotación, al señalar los atributos que un problema debe de reunir para su aplicación, y al mostrar las estrategias operativas del algoritmo básico; y se restringe la aplicación de la técnica mediante gráficas de precedencia y disponibilidad a través del tiempo. En el capítulo tercero, se aplica la ramificación y acotación con el algoritmo que resuelve el problema de itineración de actividades de igual tiempo de ejecución con restricciones de precedencia y perfil. En el capítulo cuarto, se presentan las experiencias de la elaboración del programa de computadora, iniciando con la forma en que se realizaron las estructuras de datos; se muestra la complejidad de la creación de combinaciones factibles de elementos de una gráfica de precedencia; y se concluye al presentar tablas comparativas de la eficiencia entre el algoritmo de ramificación y acotación y el de programación dinámica. En el apéndice A se presenta la codificación del programa, el cual se elaboró en el lenguaje Pascal dentro del Editor de Turbo Pascal versión 5.5 de Microsoft en una computadora compatible AT 80286. Se asume que algunos procedimientos o funciones pueden ser aprovechadas en la resolución de algún problema combinatorio. El apéndice B contiene la terminología más usada de la teoría de itineración, el cual tiene como objetivo familiarizar al lector con todos los significados y sinónimos, como apoyo para futuras investigaciones.

CAPITULO 1

TEORIA DE LA ITINERACION

El control detallado de operaciones de las industrias se ha convertido recientemente en el objetivo de los investigadores. Un aspecto importante del control industrial, es la decisión del uso eficiente de las máquinas en cada instante de tiempo. Muchos factores se tienen que tomar en cuenta para hacerlas, como por ejemplo, la disponibilidad de los recursos, costos de ejecución, tiempos de vencimiento, etc. Es este tipo de decisiones lo que se denomina itineración.

1.1 Descripción y objetivos.

De una manera muy general, podemos considerar que la itineración es la asignación de recursos en el tiempo para realizar un conjunto de tareas. A partir de esta definición, se puede establecer una división específica del estudio de la itineración:

a) Considerada como una función para tomar decisiones, que permite determinar itineraciones.

b) Considerada como un cuerpo teórico: es un conjunto de principios, modelos, técnicas y conclusiones lógicas que proporcionan una visión de la función de itineración.

El problema práctico de la asignación de recursos en el tiempo para realizar un conjunto de trabajos, se encuentra en una gran variedad de situaciones. Sin embargo, en la mayoría de los casos, la itineración no es considerada hasta que algunos problemas fundamentales de la planeación están resueltos, reconociendo que las decisiones itinerarias son de importancia secundaria dentro de las decisiones gerenciales. Por ejemplo, en las aplicaciones de manufactura, la pregunta fundamental se refiere a la selección del producto a ser manufacturado y determinación de la escala de producción. Después de que estudios de mercado y análisis económicos son realizados para determinarlo, la atención de la planeación tecnológica se centra en la manera en que se debe manufacturar el producto. Solamente después de que se han contestado estas preguntas, y la disponibilidad de recursos es conocida, se vuelve necesario considerar los problemas de itineración. La determinación de las respuestas es la función de planeación; en contraste, la función de itineración asume que las

respuestas a estas preguntas ya existen. Por lo tanto, la función de itineración se vuelve relevante en la situación donde la naturaleza de los trabajos a ser itinerados ha sido descrita y la configuración de los recursos disponibles ha sido determinada.

Por otro lado, la teoría de la itineración se ocupa principalmente de los modelos matemáticos que se relacionan con la función de itineración, y de las técnicas que proporcionan la solución a los diferentes problemas. La perspectiva teórica es predominantemente una aproximación cuantitativa, que intenta capturar la estructura del problema en forma matemática y concisa. En particular, esta aproximación comienza con la traducción de los objetivos y restricciones de decisión a funciones objetivo y restricciones en forma explícita.

La función objetivo debería de consistir de todos los costos del sistema que dependen de las decisiones de itineración. Sin embargo, en la práctica tales costos son a menudo difíciles de medir, o de identificar completamente. Existen tres tipos de objetivos que prevalecen en la itineración: utilización eficiente de los recursos, respuesta rápida a las demandas y terminación cercana a la fecha de los contratos establecidos.

Las restricciones de factibilidad que comunmente se encuentran en los problemas de itineración, se refieren a los límites que existen en la capacidad de los recursos disponibles, y al orden de precedencia en la ejecución de los trabajos. Una solución a un problema de itineración es cualquier solución factible de estas dos clases de restricciones.

Debido a que el desarrollo de la teoría de itineración, está motivado para resolver los problemas del área de manufactura industrial, se utiliza frecuentemente la terminología de ésta para describir los problemas de itineración, en ella los recursos son llamados máquinas y los módulos de actividades básicas son llamados trabajos. Algunas veces los trabajos pueden consistir de varias actividades elementales, llamadas operaciones, que están relacionadas con restricciones de precedencia. En el campo computacional por ejemplo, las máquinas pueden representar almacenamiento primario o secundario; mecanismos de entrada o salida; o subrutinas de librería. Mientras que los trabajos pueden ser archivos, tareas de impresión o transmisión, o programas, todos en espera de ser almacenados, leídos, transmitidos, o ejecutados.

En la itineración de un conjunto de trabajos de un sistema de producción, el objetivo es cerciorarse de que el orden tecnológico

prescrito de cada trabajo sea satisfecho y que no haya conflictos en su ejecución. La situación ideal dentro del sistema de producción es que las máquinas estén ocupadas continuamente, en esta forma se evita el ocio; la manufactura sea suficiente para el trabajo; los inventarios de materiales, trabajos en espera, y productos terminados, estén en un nivel razonable, de esta forma se eviten las crisis en los inventarios; y que se cumpla con los tiempos de terminación y entrega, previamente establecidos, de tal manera que cumplen con los planes elaborados.

Desde un punto de vista práctico, los modelos de itineración están muy restringidos, ya que no consideran factores importantes que se presentan en cualquier sistema de producción. Como por ejemplo, podemos considerar el flujo de producción interrumpido frecuentemente por eventos impredecibles, causando retrasos. Estos retrasos pueden ser debido a las siguientes causas:

a) La probabilidad natural de falla de máquina.
b) La manera diferente de trabajar de los operadores.
c) El ausentismo de los trabajadores.
d) El retraso en el reemplazo de materiales, instalaciones o herramientas.
e) Los cambios que pueden ser hechos en la especificación de trabajos y los vencimientos asociados.
f) La presión que un cliente puede ejercer para su trabajo, que puede retrasar a otros.
g) El rechazo de algunas unidades defectuosas que pueden causar que se repitan las operaciones.
h) La variación de los tiempos de proceso que depende de los métodos de estimación empleados.

1.2 Estructura básica de los modelos de itineración.

La actividad de un sistema de producción se inicia generalmente por la recepción de ordenes del cliente. La orden puede comprender uno o varios trabajos, cada uno de ellos debe ser planeado antes de ser realizado, esto incluye establecer el orden tecnológico en el cual los trabajos van a ser procesados en las máquinas; especificar el material, instalaciones y herramientas que son requeridas; determinar el tamaño económico de producción del trabajo, si existe; estimar el tiempo de proceso, incluyendo transportación y tiempo de máquina, para cada operación que comprende el trabajo; encontrar el tiempo límite en el cual el trabajo debe ser completado en respuesta a un tiempo

deliberadamente dictado por el cliente, o por el reabastecimiento del inventario. Cuando el trabajo es iniciado se envía a través de los centros de trabajo para ejecutar las operaciones asociadas.

En forma específica los elementos del sistema de producción se describen a continuación: una máquina del sistema de producción está equipada con varios grupos de máquinas. Cada grupo está constituído únicamente por máquinas idénticas con respecto a sus habilidades de procesamiento del trabajo. La cantidad de trabajo constituyen varias actividades en espera de ser realizadas. Un trabajo puede estar constituído, por una unidad simple o un conjunto idéntico de unidades, generalmente esto está determinado por una razón económica. Cada trabajo requiere una o más operaciones, y cada una de éstas será procesada en una máquina simple. Una operación está determinada enteramente por la especificación del trabajo y la máquina correspondiente. Las operaciones que comprenden cada trabajo son procesadas en máquinas con una ruta específica, que se conoce como orden de máquina. Este está impuesto generalmente por los requerimientos tecnológicos que siempre dicta el proceso básico de manufactura, y el tipo de máquina, instalaciones y herramientas que serán usadas. Cada operación tiene un tiempo de proceso asociado, en el que se completa la ejecución de dicha operación. En él se contemplan los siguientes aspectos:

a) El tiempo de transportación de trabajo, es el requerido para transportar el trabajo desde una máquina a otra, de acuerdo al orden de máquina preespecificado.
b) El tiempo de disposición de máquina, es el requerido en preparar la máquina para ejecutar otra operación.
c) El tiempo de corrida de operación, es el requerido para ejecutar el trabajo en una máquina específica.
d) El tiempo de demolición de máquina, es el requerido para regresar a su estado original a la máquina, después de que la operación ha sido completada.

Al igual que todos los modelos que tratan de representar a un fenómeno de la realidad, los modelos de itineración están basados en ciertas suposiciones que tratan de restringir el comportamiento del fenómeno, para así poder representarlo y obtener una posible solución. Los supuestos que se consideran se refieren a los trabajos, máquinas y al tiempo de procesamiento. Estos supuestos son los siguientes:

a) Supuestos referentes a cada trabajo.

a.1) Es procesado de acuerdo a un orden de máquina preespecificado, y no se permite uno alternativo.
a.2) Una vez comenzado, debe ser completado, esto significa que no ocurren cancelaciones.
a.3) Una vez comenzado en una máquina debe ser ejecutado hasta terminarlo, antes que otro trabajo pueda comenzar en la misma máquina.
a.4) Es una entidad, aún cuando el trabajo está compuesto de unidades individuales. Esto elimina división de trabajos entre dos o más máquinas.
a.5) No puede ser procesado por más de una máquina al mismo tiempo.
a.6) Tiene que esperar a que una máquina esté disponible, y por lo tanto el inventario de espera es permitido.
a.7) Tiene un cierto número de operaciones, cada una de ellas puede ser ejecutada por sólo una máquina.
a.8) Puede tener un vencimiento o fecha de terminación, que es generalmente determinada por el cliente y la gerencia del sistema de producción. Tal vencimiento, si existe, es fijo.
a.9) Puede ser procesado más de una vez en cualquier máquina.

b) Supuestos referentes a cada máquina.

b.1) Existe sólo una de cada tipo en el sistema de producción.
b.2) Dentro del sistema de producción opera independientemente, y así cada una es capaz de operar a su máxima tasa de producción.
b.3) Está continuamente disponible para asignación durante el período de itineración considerado, sin ninguna interrupción por falla o mantenimiento.
b.4) Puede procesar a lo más un trabajo en un mismo tiempo.

c) Supuestos referentes a los tiempos de procesamiento.

c.1) Son conocidos y finitos.
c.2) Pueden incluir implícitamente, el tiempo de transportación entre máquinas, y tiempos de readaptación y remodelación de máquinas, si existen.
c.3) Si incluyen cambios de tiempo, son independientes de la máquina en donde los trabajos son ejecutados.

Los modelos de itineración pueden relajar alguno de estos supuestos, convirtiéndose en un poco más realistas, sin embargo, conforme se vaya haciendo ésto, la resolución se vuelve cada vez más complicada. Para los modelos que no consideran una gran cantidad de estos supuestos, se utiliza como técnica de resolución

a "la simulación", para mayor información de esta técnica se recomienda consultar [3].

1.3 Clasificación de los modelos de itineración.

En la teoría de itineración generalmente sus modelos son clasificados de acuerdo a una de las siguientes características:

a) La cantidad de máquinas que ejecutan los procesos, los que pueden ser denominados como procesos con máquina simple o proceso con multimáquina.

El problema de secuenciación pura, es un caso particular del problema de itineración, en donde una ordenación de los trabajos determina completamente un itinerario. El más simple de los problemas de secuenciación es en donde sólo hay una máquina simple. Ese caso, puede parecer trivial o muy elemental para el análisis, sin embargo su estudio es significativo, ya que provee un contexto para investigar muchas medidas de ejecución diferentes (las que se presentan en 1.4) y varias técnicas de resolución. Es fundamental en la comprensión de los conceptos de itineración, lo que en última instancia facilitará la modelación de sistemas complicados. Frecuentemente en los sistemas muy complejos, el problema de máquina simple aparece como un componente elemental, y en algunas ocasiones es posible resolver los problemas independientemente y posteriormente incorporar el resultado a una solución general.

b) Por el tipo de patrón de flujo para cada trabajo, el cual puede ser un patrón único para cada trabajo, lo que se denomina "flow shop"; o un patrón idéntico para todos los trabajos, denominado "job shop".

Cada trabajo en un "job shop", tiene un patrón de flujo diferente, esto significa que no existe un patrón de flujo común de máquina a máquina. Sin embargo en un "flow shop" existe un patrón de flujo simple, de ese modo todos los trabajos fluyen de principio a fin. Así puede pensarse que las máquinas están en serie. En un sistema, las máquinas están numeradas en orden ascendente, empezando con la máquina 1 y finalizando con la máquina m, donde m es el número total de máquinas.

La figura 1.1 A) representa el flujo de trabajo en un "flow shop", en donde todos los trabajos requieren una operación en cada

máquina. La figura 1.1 B) representa el flujo de trabajo en un "flow shop" más general. Para el segundo caso, hay que observar que los trabajos pueden requerir menos de m operaciones, esas operaciones pueden no siempre requerir máquinas adyacentes, y que la operación inicial y terminal pueden no ser las máquinas 1 y m respectivamente.

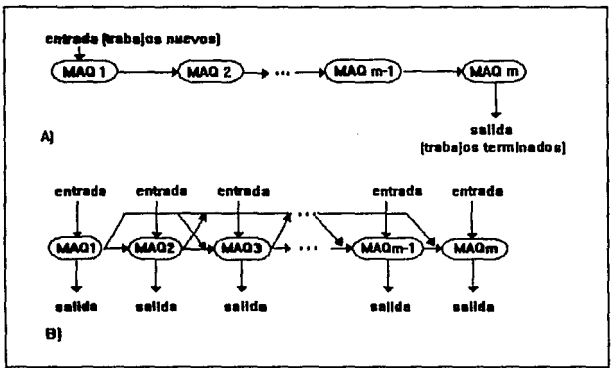


Figura 1.1 Flujos de trabajo en un flow shop.

Aunque el "flow shop" se parece a las líneas de producción, en donde todos los trabajos tienen un patrón idéntico, existen diferencias notables, que se mencionan en seguida:

i) Un "flow shop" está equipado para manejar diferentes trabajos, mientras que las líneas de producción ejecutan productos iguales.

ii) Los trabajos en un "flow shop", no tienen que ser procesados en todas las máquinas. Esto significa que un trabajo puede saltar una o más operaciones de acuerdo con su plan de elaboración. En las líneas de producción todos los trabajos se mueven de una estación

de trabajo a otra.

iii) En un "flow shop", cada máquina es independiente de las otras y pueden ser itineradas separadamente, mientras que cada estación de trabajo de una línea de producción depende de su predecesora.

iv) En un "flow shop", cada trabajo tiene su propio tiempo de procesamiento en cada máquina. Sin embargo, en una línea de producción todas las unidades de un producto tienen un tiempo estándar de ejecución en cada estación de trabajo.

Es conveniente mencionar que el problema de "flow shop" puede ser considerado como un caso especial del problema de "job shop".

c) Por el comportamiento de los arribos de los trabajos al sistema de producción, que nos proporciona la distinción entre los problemas estáticos y los dinámicos.

En un problema estático un número fijo y finito de trabajos arriban simultáneamente al sistema de producción, y están inmediatamente disponibles para ser ejecutados en máquinas que están detenidas, de esta forma el problema estático no considera en forma explícita a la variable tiempo. Por otro lado, en un problema dinámico, los trabajos están arribando intermitentemente, y por lo tanto se tiene que considerar la variable tiempo.

El problema estático es una simplificación drástica de un proceso de producción realista, y sus soluciones están restringidas a casos muy simples, y por ello tienen poco valor práctico. Pero el estudio de éstos indudablemente nos ayuda a comprender y tener una mejor visión de la solución de los problemas dinámicos.

d) Por el comportamiento de elementos del sistema de producción, el cual puede ser determinístico o estocástico.

Los modelos determinísticos están caracterizados por el hecho de que sus elementos no involucran una variación en el tiempo, y por ello una decisión puede ser determinada de manera precisa. Los modelos estocásticos están caracterizados por un cambio explícito en el tiempo, y por la incertidumbre del cambio de los elementos, en este sentido los cambios son estudiados por medio de estadísticas.

e) Por el tiempo de resolución del problema mediante el uso de

una computadora, el cual puede ser NP-completo (No Polinomial Completo) o no NP-completo.

Un problema se dice polinomial, si existe un algoritmo en el cual el tiempo requerido para su resolución, está acotado por una función polinomial del tamaño del problema. Donde el tamaño del problema se entiende como la longitud de un código (v.gr. binario) de los datos del problema. Por ejemplo, si se tiene una gráfica G = [X,A] con N igual al número de nodos del conjunto X y M igual al número de arcos del conjunto A, entonces una ruta más corta se encuentra a lo más en un tiempo O(NM), un flujo máximo en un tiempo $O(N^3)$, y un árbol de peso mínimo en un tiempo $O(N^4)$.

Un algoritmo se dice no determinístico, si además de las afirmaciones usuales que permiten una selección, contiene a otras afirmaciones. La clase de problemas que se pueden resolver en tiempo polinomial por algoritmos no determinísticos se le denomina clase NP, esto significa que, si para cada instrucción seleccionada, la selección es correcta, entonces el tiempo de computadora es polinomial. Por otro lado, si se enumeran las posibles selecciones el algoritmo se vuelve determinístico, pero con tiempo de computadora exponencial. Un problema NP-completo es aquel para el cual las únicas soluciones conocidas consisten, en esencia, en intentar todas las posibilidades.

Los problemas NP-completo incluyen muchos problemas muy conocidos, dentro de los que podemos mencionar los siguientes: del agente viajero, de la mochila, de itineración de "n" trabajos en "m" máquinas, de itineración de actividades de igual tiempo de ejecución con restricciones de precedencia y tiempo, etc.

1.4 Criterios de optimización.

La resolución de un problema de itineración, requiere como parte fundamental, el establecer el o los criterios de optimización que se van a utilizar. En muchas ocasiones ésto no es fácil, debido a que, si se desea optimizar más de uno, como podría ser el minimizar el tiempo y el costo, resulta conflictivo hacerlo simultáneamente, ya que existe una relación entre ambos, y se hace necesario decidir cuál es más importante y si únicamente se optimiza uno o si se optimizan ambos. Existen algunos métodos que contemplan la optimización de varios criterios, algunos de ellos lo hacen a través de factores de peso, que determinan la importancia de alguno con respecto a los demás.

Dentro de los criterios que más se utilizan en la actualidad, se tienen los siguientes:

a) De tiempo de terminación.
b) De tiempo de flujo.
c) De vencimiento de contrato.
d) De inventario de espera.
e) De utilización de máquinas.
f) De cambio de máquinas.

a) Criterio de tiempo de terminación

Este es el que más frecuentemente se utiliza, además de que es el más simple y fácil de medir. En él, se considera la minimización del tiempo de terminación de los trabajos efectuados en el sistema de producción.

El tiempo de terminación de un trabajo está definido de la siguiente manera:

$$C_j = R_j + \sum_{m=1}^{M} t_{jm} + \sum_{m=1}^{M} W_{jm} = R_j + T_j + W_j$$

Donde:
- C_j -representa el tiempo de terminación del trabajo j.
- R_j -representa el tiempo de preparación del trabajo j.
- t_{jm} -representa el tiempo de procesamiento del trabajo j en la máquina m.
- W_{jm} -representa el tiempo de espera del trabajo j antes de ser procesado en la máquina m.

Esto significa, que el tiempo de terminación es igual al tiempo en el que el trabajo es preparado (R_j) más el tiempo total de procesamiento en todas las máquinas (T_j) más el tiempo total de espera para entrar a cada máquina (W_j).

En general, los tiempos de preparación y procesamiento son conocidos de antemano, en este caso C_j dependería únicamente de W_j. Y entonces si el tiempo total de espera es mínimo, la secuencia es óptima con respecto al tiempo de itinerario.

Este criterio se puede extender a un conjunto de trabajos, en tal caso el tiempo de terminación es el máximo tiempo de terminación de alguno de los trabajos, de esta forma se obtiene lo

que se conoce como "tiempo de itinerario" [1], el que matemáticamente se obtiene de la siguiente manera:

$$C' = \max_j \{ C_j \}$$

Otra manera alternativa de medir el tiempo de terminación, es el utilizar las medias, las cuales se obtienen dividiendo entre el número total de trabajos:

$$\overline{C} = \overline{R} + \overline{T} + \overline{W}$$

Y nuevamente como R y T son constantes, el itinerario que es óptimo con respecto a C es también óptimo para W. Puede suceder que dos secuencias cualesquiera tengan el mismo tiempo de itinerario, pero su correspondiente tiempo medio de terminación sea diferente.

b) Criterio de tiempo de flujo.

Este criterio utiliza la minimización del tiempo de flujo de los trabajos. El tiempo de flujo, se define como el tiempo que un trabajo ocupa en el sistema de producción, el que matemáticamente se expresa como:

$$F_j = T_j + W_j \qquad \text{ó} \qquad F_j = C_j - R_j$$

Y de la misma manera que con el criterio anterior, se pueden utilizar las medias como medida alternativa:

$$\overline{F} = \overline{T} + \overline{W} \qquad \text{ó} \qquad \overline{F} = \overline{C} - \overline{R}$$

Es importante señalar que cuando el tiempo en el que empieza un trabajo (R_i) es igual a cero, el tiempo de flujo de un trabajo se vuelve idéntico a su tiempo de terminación, y se puede aplicar indiferentemente el criterio de tiempo de terminación o el criterio de tiempo de flujo.

c) Criterio de vencimiento de contrato.

Cuando se tiene preestablecido un vencimiento, el criterio sobre éste predomina sobre cualquier otro, ya que las reacciones del cliente son irremediables, cuando las promesas no son

---

[1] Se recomienda ver el apéndice B

cumplidas. En muchas ocasiones, se tiene una penalización contractual por cada día de retraso de la fecha de entrega, y por ello la gerencia trata de minimizar estos costos o eliminarlos.

Los costos de penalización debido al retraso en la entrega del trabajo incluyen los siguientes aspectos:

i) Tiempo gastado en el trato directo con el cliente, ya sea por correo o por llamadas telefónicas.
ii) Cláusula de penalización en el contrato, si existe.
iii) Pérdida del prestigio y posiblemente pérdida del cliente.
iv) Apresuramiento debido a que los trabajos deben de ser realizados más rápidamente con un costo extra en la preparación de las máquinas y un uso ineficiente del equipo y manufactura.

Existen tres maneras de medir la diferencia entre el tiempo de vencimiento preestablecido y el tiempo real de terminación del trabajo :

i) Considerando que el tiempo de terminación de un trabajo falla en coincidir con su respectivo vencimiento. Este se calcula de la siguiente manera:

$$L_j = C_j - D_j \quad \text{ó} \quad \bar{L} = \bar{C} - \bar{D}$$

Donde: D_j -denota el tiempo de vencimiento contratado.

ii) Considerando únicamente cuando la diferencia entre el tiempo de terminación de un trabajo y su vencimiento es positiva, esto significa que el trabajo se termina después de su vencimiento acordado, y se tiene entonces un trabajo retrasado. Este se calcula de la siguiente manera:

$$T_j = \max \{0, L_j\}$$

iii) Considerando únicamente cuando la diferencia entre el tiempo de terminación de un trabajo y su vencimiento es negativo, esto significa que el trabajo se termina antes de su vencimiento acordado, y se tiene entonces un trabajo adelantado. Este se calcula de la siguiente manera:

$$E_j = \max \{ 0, -L_j \}$$

En general, el criterio del trabajo adelantado no es muy

utilizado, ya que los trabajos retrasados son frecuentemente el principal interés, debido a las penalizaciones que pueden acarrear. Sin embargo, los trabajos adelantados pueden acarrear penalidades internas, debido a los efectos de almacenaje o para los planes futuros.

d) Criterio de inventario de espera.

Cuando se tienen más de un trabajo en el sistema de producción, cada uno puede competir en cualquier momento por una máquina en particular. Esto dá como resultado un inventario de espera que está conformado por los trabajos que hacen cola antes de cada máquina, esperando ser procesados. El inventario de espera puede ser medido con el número total de trabajos en el sistema de producción N_t, formado por el número de trabajos que están siendo procesados N_p, y el número de trabajos esperando ser procesados N_q.

e) Criterio de utilización de máquina.

La cantidad y naturaleza de las labores a ser realizadas en el sistema de producción y el incremento de la complejidad del sistema de producción, por sí mismo, ocasiona frecuentemente que las máquinas queden ociosas durante varios períodos de tiempo. Como consecuencia, la capacidad productiva del sistema es subaprovechada. Es por la importancia de la utilización de las máquinas, que este criterio se basa en la minimización del tiempo total de ocio de cada máquina, el que se calcula de la siguiente manera:

$$I = \sum_{k=1}^{M} I_k$$

Donde: I_k - denota el tiempo total de ocio de la máquina k

f) Criterio de cambio de máquinas.

Este criterio utiliza la minimización del tiempo empleado en readaptar y preparar una máquina para realizar otra operación. Es importante considerar este tiempo ya que denota la capacidad productiva de la máquina y la utilización de todo el sistema de producción. Para este caso se considera al tiempo total de cambios de todas las máquinas que está formado por el tiempo que cada máquina emplea para readaptarse en la realización de todos los trabajos que ejecuta.

Criterios múltiples.

Frecuentemente en la realidad, es necesario optimizar con respecto de varios criterios, los cuales en la mayoría de las veces están en conflicto. Para tal efecto existen tres aproximaciones, ideadas por S.E. Elmaghraby[1] , que actualmente son usadas:

a) Aproximación de la evaluación compositiva.

Esta es la aproximación más evidente de las tres, consiste en utilizar pesos para los diferentes criterios y así obtener uno general. Estos pesos son generalmente costos. Sin embargo, tal procedimiento se encuentra con el mismo problema que cuando se consideran los criterios por separado, esto es ¿qué peso asociar a cada criterio?. Si se utilizan pesos inapropiados con los criterios, el beneficio del criterio múltiple puede ser en esencia neutralizado.

b) Aproximación de la evaluación secuencial.

Esta aproximación se utiliza cuando se tiene un orden de prioridad con cada criterio. Se evalua de una manera tal que cada criterio es tratado individualmente. Si el criterio primario tiene una solución única, el proceso se termina. Sin embargo, si tiene más de una solución, el criterio secundario es evaluado entre las soluciones obtenidas. El procedimiento continua hasta que una solución óptima de un subconjunto de criterios o de todos los criterios sea agotada.

c) Aproximación de la evaluación restringida.

De manera similar a la anterior, se selecciona un criterio como el principal, pero considera a los otros criterios de diferente manera. Esencialmente, el procedimiento consiste en optimizar con respecto al criterio principal retringido con los otros criterios, que forman el espacio solución. Esta aproximación se usa frecuentemente cuando sólo se consideran dos criterios.

Es conveniente puntualizar que los problemas reales que se encuentran representados por los modelos de itineración exigen que sus soluciones sean enteras, esto implica que las variables de decisión tienen sentido sólo si adquieren valores enteros. Esta

---

[1] Presentadas en [2]

restricción es difícil de manejar en forma matemática, sin embargo se han hecho algunos progresos en el desarrollo de procedimientos de solución para el caso de los problemas de programación lineal, sujetos a la restricción adicional de que las variables de decisión (todas o algunas) deben tener valores enteros.

El origen de los métodos de solución del problema de programación entera se remonta al año de 1958, cuando se propuso una técnica para generar soluciones enteras usando programación lineal. Como consecuencia de esta idea se desarrollaron los siguientes métodos:

a) Método de planos de corte.
b) Método de la programación dinámica.
c) Técnica de ramificación y acotación.

El método de planos de corte, es poco eficiente computacionalmente y requiere demasiadas iteraciones. Debido a estas desventajas este método no se utiliza para resolver problemas grandes de programación entera.

El método de programación dinámica es una estrategia de enumeración implícita muy flexible, que puede ser aplicada directamente a la mayoría de los problemas de secuenciación de una máquina simple. Sin embargo no existen opciones de diseño importantes que involucran la aplicación de la técnica a una clase específica de problemas. Porque las demandas computacionales de la programación dinámica crecen exponencialmente conforme el tamaño del problema aumenta.

La técnica de ramificación y acotación, surgida hace dos décadas, ha resultado ser una de las mejores herramientas prácticas para la solución de problemas de optimización discreta. Su atractivo radica en la habilidad de eliminar implícitamente grupos grandes de soluciones potenciales, sin evaluarlos explícitamente. A semejanza de la programación dinámica, esta técnica es una estrategia, y como tal se debe combinar con la estructura del problema específico que se desea resolver, formando así un algoritmo de solución adecuado.

La ramificación y acotación podría ser la mejor ilustración de cómo un desarrollo en computadora, de una técnica matemática, puede requerir una buena cantidad de juicio personal. La lista de opciones tácticas en donde este juicio puede ser ejercido, incluye la elección del cálculo de la cota inferior, la utilización potencial de una solución inicial, la incorporación de

verificaciones complicadas de dominancia, y la especificación de mecanismos de ramificación. A pesar de la existencia de estas opciones, y el hecho de que ellas no pueden ser evaluadas independientemente, la técnica ha sido aplicada con éxito a una gran variedad de problemas.

En el siguiente capítulo, se presenta una descripción del enfoque clásico de la ramificación y acotación, y se define un tipo especial de restricciones y estrategias operativas, las cuales pertenecen al problema de itineración de trabajos de igual tiempo de ejecución con restricciones de precedencia y perfil.

CAPITULO 2

DESCRIPCION Y DESARROLLO DE LA TECNICA DE RAMIFICACION Y ACOTACION

Los algoritmos de ramificación y acotación de enumeración implícita han surgido recientemente como una técnica para encontrar soluciones óptimas de problemas de óptimización discreta. Las aplicaciones de la técnica han crecido tan rápidamente que una lista de referencias excedería varios cientos. Ejemplos representativos de éstas incluyen los problemas de: secuenciación de "flow shop" y de "job shop", del agente viajero, de asignación cuadrática general, de programación entera y de itineración de actividades de igual tiempo de ejecución con restricciones de precedencia y perfil.

2.1 Enfoque clásico de ramificación y acotación.

En esta técnica, el conjunto de soluciones factibles se particiona en subconjuntos más simples (Esto es lo que se debería de hacer en la práctica, si uno está buscando por ejemplo, una aguja en un pajar. El pajar es grande, y es imposible buscar en todo simultáneamente, así que se puede dividir visualmente en lado izquierdo y lado derecho, y seleccionar uno de ellos para buscar la aguja, manteniendo el otro en espera, para buscar después si es necesario). A continuación un subconjunto prometedor se selecciona, para encontrar la mejor solución factible, y se almacena esta información, en algunos casos podría darse por terminada la búsqueda. Se particiona nuevamente, el subconjunto y se repite el mismo proceso, hasta encontrar que la solución almacenada es mejor que la recien obtenida.

La técnica en la optimización discreta, ha sido aceptada debido a que es conceptualmente simple y fácil de entender; además de que ajustable para su implantación computacional; y generalmente los métodos alternativos no están disponibles. La estrategia de aplicación, se basa en la premisa de que el problema que se va a resolver cumple con las siguientes propiedades:

a) Naturaleza combinatoria. Un problema combinatorio está formado por un conjunto finito y numerable de objetos *P*, en donde cada uno se selecciona mediante un parámetro de admisión *x*, generando subconjuntos. La solución del mismo, establece selecciones de r objetos de entre todos los disponibles, precindiendo de cómo éstos puedan ordenarse, y se evalua con una

función real de costo $f(p,x)$.

b) Ramificalidad. Esto significa que se tiene un conjunto finito y contable de todas las soluciones del problema, el cual se particiona recursivamente en subconjuntos disjuntos.

c) Racionalidad. Esto significa que cada solución tiene un único valor calculado de los valores de los objetos. Una solución óptima global para el parámetro x, es una solución s^*, tal que $f(s^*,x) \leq f(s,x)$, para todo s que pertenece al espacio solución S.

d) Acotabilidad. Implica la obtención de una estimación del valor de la mejor solución contenida en cualquier conjunto de soluciones, en donde el valor actual de la mejor solución en el conjunto es inferior o igual a la estimación.

El concepto de ramificación y acotación explota éstas propiedades para poder implícitamente y explícitamente construir un árbol que describa todas las operaciones realizadas, y efectuar una búsqueda para encontrar la mejor solución.

El proceso de ramificación puede representarse por medio de un árbol de búsqueda, donde cada nodo del árbol está asociado a un subconjunto de T, el que representa al conjunto de todas las soluciones. Sea T_i un subconjunto de T, esto es, una colección de soluciones. La ramificación es el proceso de particionar un subconjunto T_i en "m" subconjuntos disjuntos $v_1, v_2, \ldots, v_m$ donde:

$v_1 \cup v_2 \cup \ldots \cup v_m = T_i$ y $v_i \cap v_j = 0$ con i diferente de j.

El proceso de ramificación se puede visualizar como la creación ordenada de un árbol, donde el nodo inicial representa a T, y los nodos restantes representan subconjuntos T_i de T. A cada nodo N se le asocia un subconjunto T_i. Un "nodo intermedio" es un nodo N en el cual no ha habido ramificación. Un "nodo final" es un nodo intermedio N para el cual $T_i(N)$ consiste de una solución única.

En el árbol de la figura 2.1 se muestran estos conceptos para el caso m = 2. El nodo N=1 corresponde a T, el conjunto de todas las soluciones. Los nodos 2 y 3 están asociados con los conjuntos ajenos T_1 y T_2 de T.

La función de acotación inferior L, asigna a cada solución parcial s' un número real $L(s')$, lo que representa una cota inferior de costo para todas las soluciones derivadas de s'. Esta función tiene las siguientes propiedades:

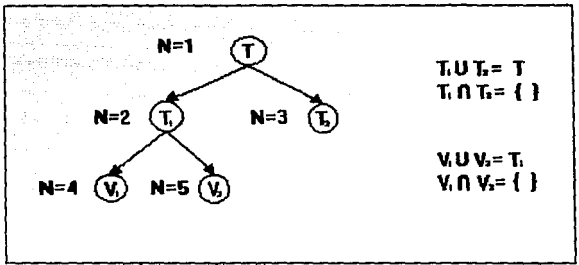


Figura 2.1 Arbol de ramificación.

a) Si s_1 es un descendiente de s_1, entonces $L(s_1) \geq L(s_1)$.
b) Para cada solución completa s, $L(s) = f(s,x)$.

Cuando una función de acotación, calcula una cota inferior $L(s_1)$ muy cercana al costo real $f(s_1,x)$ de las soluciones s_1, genbradas por s_1, se dice que dicha función es fuerte.

Una cota superior U es el costo $f(s,x)$ de alguna solución completa s, conocida al inicio. Si no existe ninguna solución conocida, se asume que U tiene un costo más grande que cualquiera posible. Cuando U está muy cercana al costo de la solución óptima, se dice que es una buena cota superior.

La forma más general de describir la técnica, es el siguiente algoritmo. El propósito es encontrar la solución óptima de un problema de optimización discreta. En este problema se busca minimizar la función objetivo.

PASO 0. Se inicia con la elección de un subconjunto del conjunto de todas las soluciones factibles del problema en cuestión, formando el primer nodo del árbol.

PASO 1. Se procede a ramificar los nodos hacia nodos nuevos, utilizando alguna regla de ramificación.

PASO 2. Se determinan las cotas inferiores para los nodos nuevos. Para cada nodo nuevo N se obtiene una cota inferior L(N) sobre el valor de la función objetivo para las soluciones factibles del nodo.

PASO 3. Se selecciona un nodo intermedio desde el cual se ramifica. Cada nodo nuevo se excluye o se sondea, si: Se encuentra que el nodo no tiene soluciones factibles, o se ha identificado la mejor solución en el nodo (de tal forma que L(N) corresponde al valor de su función objetivo).

PASO 4. Reconocer cuando un nodo final contiene una solución óptima. El proceso termina cuando no existen nodos restantes (no sondeados) y la solución de apoyo actual es la óptima. Si no existe tal solución de apoyo, entonces el problema no tiene soluciones factibles. De otra manera, se regresa al paso 1.

Cuando se trata de maximizar la función objetivo, las modificaciones que se tienen que realizar en este algoritmo únicamente consisten en invertir los papeles de las cotas e invertir las direcciones de las desigualdades.

2.2 Estrategias operativas en el algoritmo básico.

La regla de selección, es usada para escoger el próximo nodo de ramificación del conjunto de nodos actualmente activos. Un nodo está actualmente activo durante la ejecución del algoritmo si y sólo si, ha sido generado pero aún no ha sido eliminado o ramificado. Los hijos de un nodo ramificado son generados en orden lexicográfico. El hijo es añadido al conjunto de nodos actualmente activos si:

a) Tiene alguna terminación factible.
b) No es eliminado por alguna regla de eliminación.
c) No se pierde debido a que el conjunto activo exceda el máximo tamaño preespecificado.

Para describir e ilustrar algunas de las reglas de selección más comunes, se utiliza el siguiente ejemplo:

A partir de un conjunto de trabajos independientes $\{T_1, T_2, T_3\}$ con tiempos de ejecución $\tau_1=3$, $\tau_2=2$ y $\tau_3=1$, encontrar un

itinerario para ejecutar los trabajos en una máquina simple, que minimice la suma de los tiempos de terminación.

En este problema la función de costo es la suma de los tiempos de terminación, dado por el itinerario P y los tiempos de ejecución. Por ejemplo, si P=(1,2,3), si T_1 termina en el tiempo 3, T_2 en el 3+2=5, y T_3 en el 5+1=6. Consecuentemente la función de costo vale 3+5+6=14. La figura 2.2 muestra el árbol de enumeración completa.

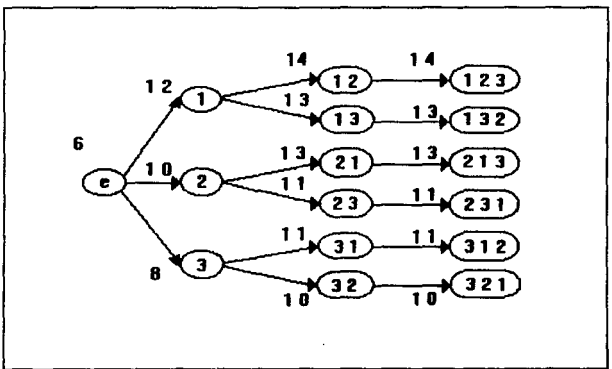


Figura 2.2 Arbol de enumeración completa.

a) Reglas de la menor cota inferior (LLB).

Se selecciona al nodo activo con la menor cota inferior de la función de costo L. En caso de empates, se puede seleccionar al nodo que fue generado primero LLB fifo o al que fue generado al último LLB lifo. En ambos casos, el orden en el que los nodos son ramificados es controlado por la función de acotación L. La figura 2.3 muestra el orden en el cual los nodos de la figura 2.2 serían seleccionados por la ramificación bajo LLB.

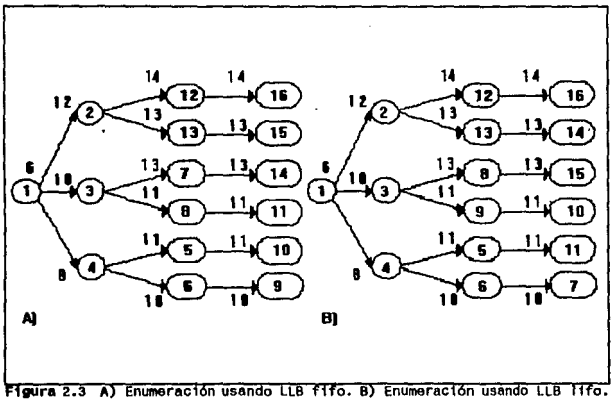


Figura 2.3 A) Enumeración usando LLB fifo. B) Enumeración usando LLB lifo.

b) Regla LIFO.

Se selecciona al nodo activo que fue generado al último, pero saltando a los nodos activos que son soluciones completas, a menos que no sean nodos activos incompletos. Esto da una búsqueda a primera profundidad, en donde el orden en el cual los nodos son seleccionados es independiente de la función de acotación. La figura 2.4 muestra el orden de generación del árbol del ejemplo.

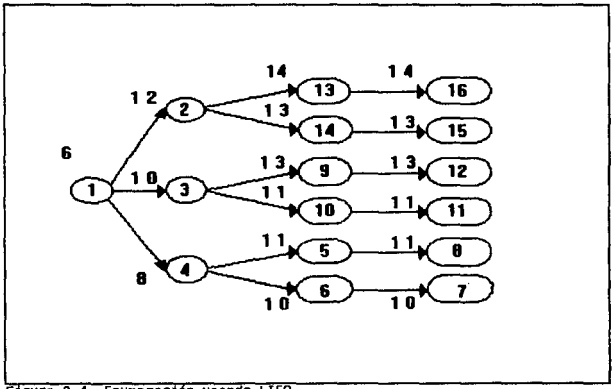


Figura 2.4 Enumeración usando LIFO.

c) Regla FIFO.

Se selecciona el nodo activo que fue generado primero. Esto es una estrategia de búsqueda de primera amplitud, y también es independiente de la función de acotación. La figura 2.5 muestra el uso de esta regla.

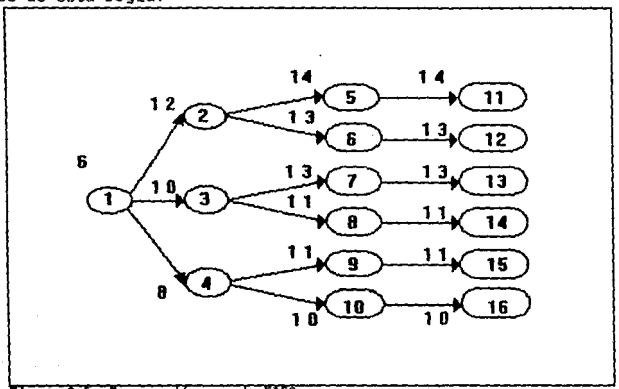


Figura 2.5 Enumeración usando FIFO.

d) Reglas de primera profundidad y mínima cota inferior (DF/LLB) ó backtrack.

Del conjunto de hijos activos generado más recientemente, se selecciona el hijo con la menor cota inferior, pero saltandose al conjunto de hijos que son soluciones completas, a menos que no haya nodos activos incompletos. En el caso de empate, se selecciona al hijo que fue generado primero DF/LLB fifo ó que fue generado al último DF/LLB lifo. En la figura 2.6 se muestra el uso de esta regla (en este caso no hay empates).

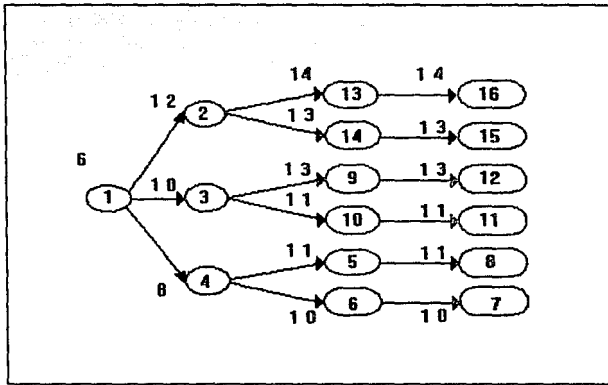


Figura 2.6 Enumeración usando DF/LLB.

En ciertas ocasiones es posible conocer una cota superior de la función de acotación, la cual puede ser usada como un dato inicial para reducir los cálculos, si ésta es cercana al óptimo.

Algunos investigadores han estudiado la relación que existe entre el uso de alguna de las anteriores reglas de selección y los requerimientos computacionales (tiempo y almacenamiento), se recomienda ver los trabajos de [20]. A continuación se presentan sus conclusiones:

a) Cuando se utiliza la regla FIFO ó la LIFO los requerimientos computacionales no se incrementarán si una fuerte función de acotación ó una buena cota superior es usada.

b) Cuando se utiliza LLB ó DF/LLB los requerimientos

computacionales pueden incrementarse, si una fuerte función de acotación es usada.

c) Cuando se utiliza LLB ó DF/LLB los requerimientos computacionales no se incrementarán, si una buena cota superior inicial es usada.

Una ayuda muy importante dentro de estas estrategias es el concepto de dominación, el cual proporciona una manera de eliminar nodos.

Dominación.
En algunas ocasiones sucede que para algún par de cojuntos "X" y "Y" contenidos en "T", la mejor solución en "X" es al menos tan buena como la mejor solución en "Y". Entonces se dice que "X" domina a "Y", y el nodo de "Y" dentro del árbol solución no necesita ser desarrollado. Si el dominio es mutuo, entonces cualquiera de los dos puede no ser desarrollado.

2.3 Las restricciones de precedencia y perfil en la ramificación y acotación.

Ahora se considera, el proceso de ramificación y acotación sujeto a las restricciones de precedencia y perfil, que están vinculadas con los problemas de itineración.

Las restricciones de precedencia se refieren a la jerarquía de selección, que los elementos tienen, es decir, la prioridad que cada elemento tiene para ser escogido antes que algún otro. Esta jerarquía, puede ser representada mediante una gráfica dirigida acíclica (árbol), en la cual, el sentido del arco denote la precedencia de los nodos que une.

Por ejemplo, considerese al conjunto X ={ a, b, c, d, e }, donde cada elemento tiene la jerarquía de selección expresada en la gráfica de la figura 2.7:

Esta gráfica muestra que "a" tiene la máxima jerarquía, debido a que se encuentra como nodo padre de la gráfica, después se tiene en una misma generación a los nodos: "b" y "c", y posteriormente se tiene la última generación formada por los nodos: "d" y "e", en donde únicamente "c" tiene jerarquía directa sobre ella, pero hay que señalar que "b" no tiene ninguna jerarquía sobre "d" o "e", e incluso sobre "c", y por ello se puede hacer la siguiente selección

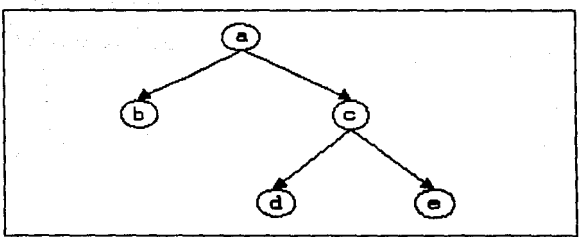


Figura 2.7 Jerarquía de selección en X.

de elementos: "a, c, d, e, b" ó "a, c, d, b, e" , ya que ninguna de ellas viola las restricciones de precedencia. Esta manera de selección puede expresarse del siguiente modo: Un nodo cualquiera puede ser seleccionado si y sólo si, todos sus nodos antecesores han sido seleccionados. A una gráfica de este tipo se le denomina **"gráfica de precedencia"**.

Comunmente, las restricciones de precedencia entre los elementos se denotan con el simbolo "<". De tal manera que $T_i<T_j$ si y sólo si, la gráfica de precedencia contiene una ruta directa desde el nodo T_i al nodo T_j. En este caso, se dice que T_i es un predecesor de T_j (o que T_j es un sucesor de T_i). Las gráficas de precedencia frecuentemente se clasifican en dos tipos:

a) In-tree, el cual es un árbol cuya característica principal es que todas las rutas conducen al nodo raíz.

b) Out-tree, el cual es un árbol cuya característica pricipal es que las rutas conducen a nodos hoja.

Existe una medida derivada de una gráfica de precedencia, la cual es usada dentro de la itineración de trabajos. Dicha medida es el nivel que cada elemento tiene en la gráfica, este nivel se calcula de la siguiente manera:

$$NIVEL(T_i) = 1 + Max\{ NIVEL(T_j) \text{ con } T_j < T_i \}$$

A esta medida también se le conoce con el nombre de "altura" del elemento. Y para el ejemplo anterior se tiene que la altura del nodo "a" es 3, para el "c" es 2 y para "b, d, e" es 1. Este concepto puede ser extendido a un conjunto de elementos "B" del modo siguiente:

$$NIVEL(B) = Max \{ NIVEL(T_i) \text{ con } T_i \in B \}$$

La restricción del perfil se refiere a la cantidad de elementos que en cada etapa de selección pueden ser escogidos, por ejemplo, si se tiene el siguiente perfil:

T_1	T_2	T_3	T_4
2	3	4	2

Este significa que en la primer etapa de selección, únicamente pueden ser escogidos dos elementos, en la segunda etapa pueden ser escogidos tres elementos, en la tercera etapa pueden ser escogidos cuatro elementos y finalmente en la cuarta etapa dos.

Un pefil es "recto", si existe el mismo número de disponibilidad para selección en todas las etapas.

Existe una medida derivada del perfil conocida como "amplitud", y se calcula al encontrar la mayor disponibilidad en las etapas, produciendo una cota superior para el perfil. Por ejemplo en el caso anterior se tiene que la amplitud del perfil es de 4, debido a la etapa 3. Se debe pensar, como algo evidente, que el perfil debe de ser lo suficientemente grande para que todos los elementos del problema puedan ser seleccionados.

Es importante señalar, que las restricciones de precedencia y perfil son mencionadas ampliamente en algunos problemas de itineración, y por lo tanto representan un mayor significado para dichos problemas, que sí se consideran por separado. Dentro de estos problemas, las restricciones de precedencia se refieren a los trabajos que se tienen que itinerar, y se encuentran representados en la gráfica de precedencia por los nodos. La restricción de perfil se refiere a la disponibilidad de procesadores que existen en un tiempo para ejecutar trabajos, y por ello a esta restricción se le conoce como restricción de perfil de tiempo. Las restricciones de precedencia de un conjunto de trabajos en conjunción con las restricciones de perfil, implican que sólo se permiten itinerarios en donde ningún trabajo comience su ejecución hasta que todos sus predecesores hayan sido terminados, de acuerdo

a la disponibilidad en cada etapa.

2.4 Estrategias operativas con restricciones de precedencia y perfil.

Las estrategias operativas de la técnica de ramificación y acotación con restricciones de precedencia y perfil, fueron desarrolladas por los doctores: Cheng-I Yang, Jia-Shung Wang y R. C. T. Lee, en el año de 1989, con el objeto de resolver un problema de itineración con esas condiciones. Ellos propusieron cinco estrategias, las cuales reducen el tamaño del árbol solución generado y el tiempo de resolución del mismo. Algunas de estas estrategias son reglas heurísticas, que recortan la generación del árbol solución y las otras reglas, sirven para cerrar las ramas del mismo.

Estrategia de los sucesores comunes.

Teorema 1. Sea $V_1 = (j_{11},\ldots,j_{1k})$ y $V_2 = (j_{21},\ldots, j_{2k})$ dos nodos del árbol solución con el mismo predecesor inmediato. Entonces se puede descartar V_2, si para cada trabajo j_{1i} $(1\le i\le k)$ en V_1, cualquier $j_{2i} = j_{1i}$ ó j_{2i} y j_{1i} tienen el mismo conjunto de sucesores inmediatos en la gráfica de precedencia.

Prueba. Se puede probar el teorema si se muestra que: "Para cualquier solución factible emanada desde V_1, debe existir una solución factible correspondiente emanada desde V_2 con el mismo número de etapas". Sea S una solución factible emanada desde V_1. Sin pérdida de generalidad, se puede asumir que en ambos nodos V_1 y V_2 los primeros r trabajos $(1\le r\le k)$ son diferentes, pero tienen el mismo conjunto de sucesores inmediatos en la gráfica de precedencia. Debido a que V_1 y V_2 están en el mismo nivel de la solución, el trabajo j_{2i} debe aparecer en alguna parte en S. Considerese el resultado de intercambiar todos los pares de j_{1i} y j_{2i} $(1\le i\le r)$. Esto genera una nueva solución S', con el mismo número de etapas, ya que cada sucesor de j_{1i} y j_{2i} $(1\le i\le r)$ en la gráfica de precedencia debe ser itinerado más tarde que j_{1i} y j_{2i} en S. Este hecho también es cierto en S'. Esto es, cada solución factible emanada desde V_1, puede transformarse en una solución factible emanada desde V_2, con el mismo número de etapas. En consecuencia, el nodo V_2 no necesita ser considerado. ■

Con base en este teorema se puede mencionar la primer regla en la estrategia.

REGLA 1. Los nodos de la gráfica de precedencia que comparten sucesores comunes deben ser recorridos sólo una vez en el árbol solución.

Para comprender un poco mejor esta regla, se desarrolla un ejemplo de itineración, que consiste del siguiente perfil de tiempo y la gráfica de precedencia de la figura 2.8.

T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8	T_9
3	2	2	2	4	5	3	2	3

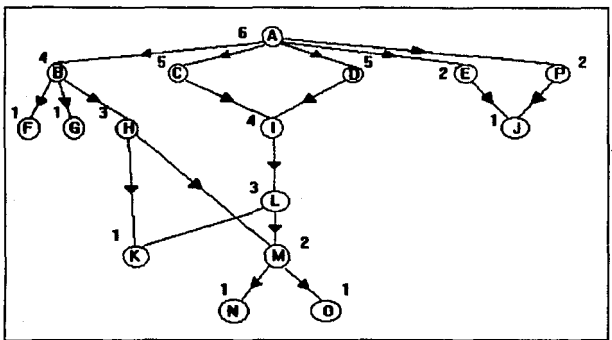


Figura 2.8 Gráfica de precedencia.

Un itinerario parcialmente desarrollado es el que se expresa en la figura 2.9. En este caso, el nodo raíz del árbol solución únicamente contiene al trabajo A. Este nodo tiene muchos sucesores inmediatos, dos de ellos son (C,E) y (D,P). Como se muestra en la gráfica de precedencia los trabajos C y D tienen el mismo sucesor inmediato, el trabajo I. De forma similar los trabajos E y P tienen el mismo sucesor inmediato, el trabajo J. Se considera ahora la regla 1, la cual señala que sólamente un nodo (C,E) ó (D,P) debe ser expandido en el árbol solución. En la figura 2.10 A) se muestra una solución factible emanada desde el nodo (C,E). Esta solución factible debe contener los trabajos D y P en algúna parte, debido a que D y P deben ser ejecutados antes que C. Ahora intercambiando C por D, al igual que E por P, se obtiene la figura 2.10 B). Al comparar la figura 2.10 A) con la figura 2.10 B), se puede observar que (C,E) es reemplazado por (D,P). Con lo que se ha transformado una solución factible emanada de (C,E), en otra solución factible emanada de (D,P), sin cambiar la longitud de la solución, es decir, el número de etapas usado para itinerar todos los trabajos. Por lo tanto se puede concluir que la longitud de una solución óptima emanada de (C,E) será la misma que una emanada de (D,P).

Estrategia de prioridad para nodos internos.

Teorema 2. Sean $V_1=(j_{11}, j_{12},\ldots,j_{1k})$ y $V_2=(j_{21},j_{22},\ldots,j_{2k})$ dos nodos en el nivel m del árbol de solución, los cuales tienen el mismo predecesor inmediato. Se puede descartar V_2 si, para cualquier i ($1\le i\le k$) uno u otro $j_{1i} = j_{2i}$ es un nodo hoja o j_{1i} es un nodo interno de la gráfica de precedencia.

Prueba. Sea S un itinerario factible emanado desde V_2. Sin pérdida de generalidad, se puede asumir que, en ambos nodos V_1 y V_2, sus primeros r ($1\le r\le k$) trabajos son diferentes. Debido a que el trabajo j_{1i} debería aparecer en cualquier parte de S, se puede intercambiar j_{1i} con j_{2i}, y obtener un nuevo itinerario S'. Ya que j_{2i} es un nodo hoja, puede ser itinerado en cualquier nivel de S. Por ello S' es una solución factible. Es claro que la longitud de S' es la misma que la de S. Esto significa que V_2 puede ser terminado. Pero el proceso inverso no siempre es cierto, debido a que pueden ser violadas las restricciones de precedencia. En consecuencia el nodo V_2 está limitado por el nodo V_1. ■

Con base en este teorema se puede mencionar la segunda regla

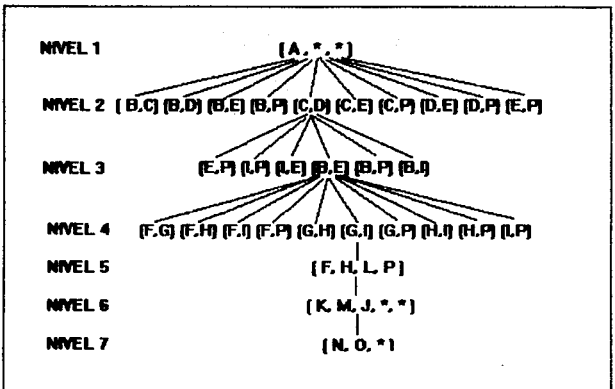


Figura 2.9 Parte del árbol solución. * indica procesadores ociosos.

en la estrategia:

REGLA 2: Los nodos internos de la gráfica de precedencia serán procesados antes que los nodos hoja.

Nuevamente para una mejor comprensión de esta regla, se considera la gráfica de precedencia anterior, pero ahora con un nuevo perfil de tiempo:

T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8	T_9
1	3	2	3	2	2	3	2	3

En la figura 2.11 se muestra el árbol solución. En este caso, se tiene que después de expandir el nodo (B,E,P), dos de los decendientes son (C,D) Y (F,J). Debido a que los trabajos C y D son

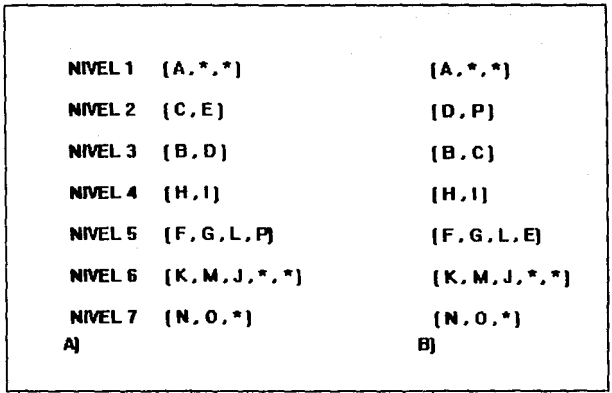


Figura 2.10 A) Solución factible emanada desde (C,E).
B) Solución factible emanada desde (D,P).

nodos internos dentro de la gráfica de precedencia mientras los trabajos F y J son nodos hoja, la regla 2 señala que el nodo (C,D) debe ser recorrido y descartar el recorrido sobre el nodo (F,J). Debido a que en cualquier solución factible emanada desde (F,J), los trabajos C y D deben aparecer, y ellos pueden ser intercambiados por F y J. La solución factible emanada desde (F,J) que es cambiada a una nueva solución factible derivada de (C,D) se muestra en la misma figura.

Esta regla sugiere que el conjunto de posibles candidatos debe ser dividido en dos subconjuntos, el de los nodos internos y el de los nodos hoja, de acuerdo con la gráfica de precedencia. Esta medida reducirá el número de combinaciones, ya que un elemento de el subconjunto de nodos hoja será escogido sólo cuando el tamaño del conjunto activo de nodos internos sea menor que el número de procesadores activos que señala el perfil de tiempo.

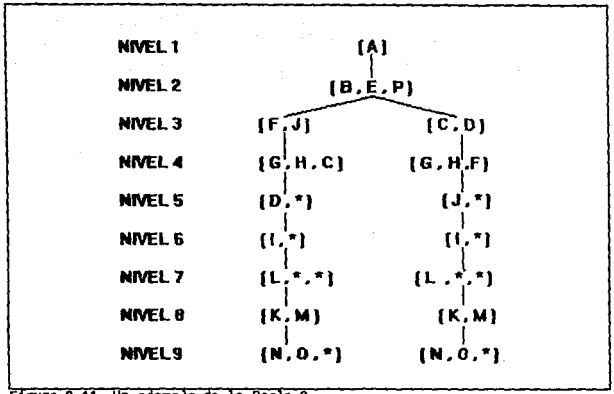


Figura 2.11 Un ejemplo de la Regla 2.

Debido a que los nodos hoja no tienen sucesores, no existe diferencia en cuanto a cómo son seleccionados para itinerarlos. Por ello se pueden escoger arbitrariamente cualquier grupo de ellos.

Como se ilustra en la figura 2.9, después de recorrer el nodo (B,E) en el nivel 3, se tienen cinco trabajos en el conjunto de actuales candidatos {F,G,H,I,P} y existen dos procesadores activos, definidos por el primer perfil de tiempo. Así se tiene que el número total de posibles combinaciones es diez,las cuales son (F,G), (F,H), (F,I), (F,P), (G,H), (G,I), (G,P), (H,I), (H,P), (I,P) y si se desarrollaran, se tendrían que eliminar siete de ellas. Pero si solamente se consideran los nodos internos de el conjunto de actuales candidatos, entonces sólo se tendrían que generar tres nodos (H,I), (H,P), (I,P). Los otros siete nodos, nunca tendrían que ser generados, y esto produce un considerable ahorro de tiempo y memoria.

Estrategia de ruta crítica.

Para poder mencionar la tercera regla, se tiene que definir una función de acotación, la que genera una cota inferior para un itinerario.

Definición: Sea V un nodo del árbol solución y PS un itinerario parcialmente desarrollado correspondiente a V. Sea A el nivel de V en el árbol solución, y sea h(PS) la altura del itinerario parcialmente desarrollado. Entonces la cota inferior del nodo V se calcula de la siguiente manera:

$$A + h(PS)$$

REGLA 3. *Sean V_1 y V_2 dos nodos del árbol solución con el mismo predecesor inmediato, entonces si V_1 tiene un itinerario parcialmente desarrollado con cota inferior menor que el de V_2, el nodo V_1 debe ser procesado antes que V_2.*

Si un nodo tiene una cota inferior menor, entonces tiene una probabilidad grande de finalizar una itineración total antes, y el último nodo de ese desarrollo, proporciona una cota superior, que cerrará otros desarrollos que tengan cotas inferiores mayores que la cota superior encontrada.

Un ejemplo de esta estrategia es el siguiente: Considerando de nueva cuenta la gráfica de precedencia anterior, se tiene que los descendientes del nodo A son B,C,D,E y P con alturas 4,5,5,2 y 2, respectivamente. Ahora se puede calcular el valor de las cotas inferiores de los dos nodos marcados con * y ** de la figura 2.12. Debido a que el nodo * está en el nivel 2 y su altura es 4 (ya que el trabajo B es el que tiene la máxima altura dentro del conjunto de trabajos no itinerados SUP={B,I,H,L,M,E,F,P,J,K,N,O,G}), La cota inferior de * es 2+4=6. De manera similar, se puede calcular el valor de la cota inferior de **, cuyo valor es 2+5=7 (debido al trabajo C dentro del conjunto SUP={C,E,F,G,H,I,J,K,L,M,N,O,P}). Si la regla 3 es aplicada, el nodo * será expandido antes que el nodo **. Como se muestra en la figura 2.12, el nodo * conduce a una solución con una cota inferior de 6, ésta se convierte en una cota superior ya que pertenece a una solución final. En este caso, tenemos que el nodo ** debe ser terminado, puesto que el valor de

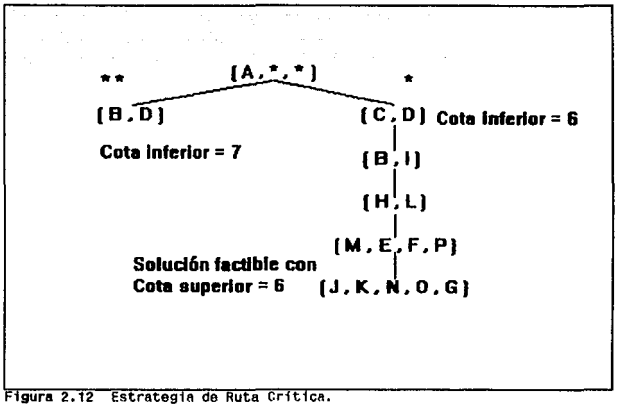


Figura 2.12 Estrategia de Ruta Crítica.

su cota inferior es 7, que es mayor a la cota superior encontrada.

Para mayor información acerca de la estrategia de ruta crítica se recomienda consultar [24,17].

Estrategia del número de trabajos itinerados.

REGLA 4. Un itinerario S no será una solución óptima si P(S,i) está contenido por P(S',i) para algún otro itinerario S'.

Esta regla se puede probar fácilmente. Si P(S,i) está contenido en P(S',i), el conjunto de trabajos que no han sido itinerados aún en la etapa i de S', es un subconjunto del conjunto de trabajos que no han sido itinerados aún en la etapa i de S. Por ello, S no puede ser terminado antes que S'. Así S, no puede ser

mejor que S'.

Un ejemplo en la aplicación de esta regla se ilustra en la figura 2.13. Los itinerarios que pasan por el nodo *** y por el nodo ** no podrán obtener mejores soluciones que el itinerario que pasa por el nodo *. Así los nodos *** y ** deben ser terminados. Se tiene que P(S",3) = {A,B,C,D,E} Y P(S,3) = {A,B,C,D,E} son iguales y por ello S" no se genera, también P(S',5)={A,B,C,D,E,F,G,H,I,P} y P(S,5)={A,B,C,D,E,F,G,H,I,P,L} donde P(S,5) contiene a P(S',5) y por ello S'no se genera.

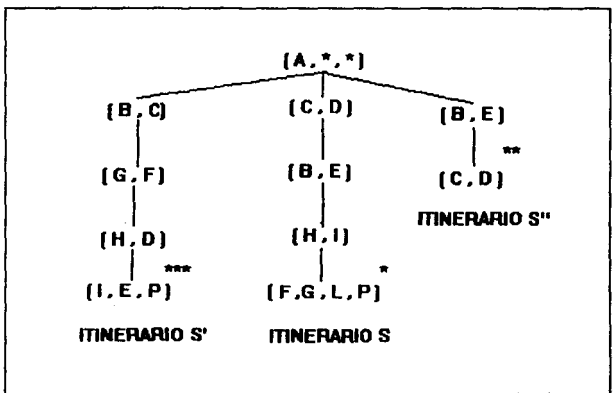


Figura 2.13 Efecto de los Trabajos Procesados.

Estrategia para procesadores ociosos acumulados.

Teorema 3. Cualquier nodo en el árbol solución con un número de procesadores ociosos acumulados (a.i) mayor que un itinerario factible será terminado.

Prueba. Supóngase que un itinerario factible toma (k+1) etapas y el número de procesadores ociosos acumulados es i. De acuerdo con el perfil de tiempo, el número total de procesadores disponibles desde la etapa 1 a la etapa k es M. Así, el número total de trabajos debería ser mayor que (M-i+1). El "+1" significa que al menos un trabajo es procesado en la etapa k+1. Si existe un nodo en el árbol solución con un número de procesadores ociosos mayor que i, entonces pasando por ese nodo, el número máximo de trabajos que pueden ser procesados antes de la etapa k+1 es (M-i-1) que es menor que (M-i+1). Por lo tanto, ramificando desde este nodo, ningún itinerario puede procesar todos los trabajos antes de la etapa k+1. Esto significa que este nodo debe ser removido del árbol solución. ■

Con base en este teorema se puede mencionar la quinta regla en la estrategia.

REGLA 5. Cualquier itinerario parcialmente desarrollado con un número de procesadores ociosos acumulados (a.i) mayor que un itinerario factible, no será mejor que esta solución factible y por ello será terminado.

El concepto de número de procesadores ociosos acumulados no sólo puede ser usado como una regla de terminación, sino que tambien puede ser usado como una regla heurística.

Cuando todas las reglas anteriores han sido usadas, y todavía existen muchos candidatos, la regla 5 puede aplicarse para seleccionar el trabajo a ser procesado en la etapa respectiva, eso maximizará el tamaño del conjunto de candidatos de la siguiente etapa. Esta modificación de la regla 5 genera la siguiente regla:

REGLA 5'. Si V_1 y V_2 son dos nodos en el árbol solución que tienen el mismo predecesor inmediato, y si el conjunto de candidatos de V_1 es mayor que el de V_2, entonces V_1 debe ser procesado antes que V_2.

Un ejemplo de esta regla se muestra en la figura 2.14, en donde hay una solución factible emanada desde el nodo raíz hasta el nodo (J,*,*), que tiene como número total de procesadores ociosos acumulados al 3. Ya que el número total de procesadores ociosos acumulados del nodo (K,M,J,*,*) es 4, éste nodo debe ser terminado.

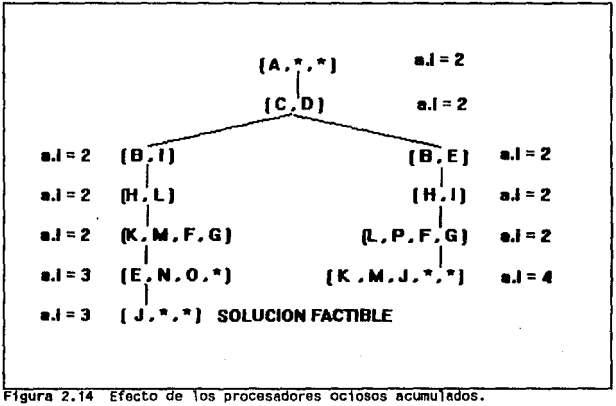


Figura 2.14 Efecto de los procesadores ociosos acumulados.

CAPITULO 3

APLICACION DE LA TECNICA DE RAMIFICACION Y ACOTACION A UN PROBLEMA DE ITINERACION

Los algoritmos de ramificación y acotación si bien son más eficientes que una enumeración completa, tienen requerimientos computacionales que frecuentemente crecen de manera exponencial o como un polinomio de alto grado en un problema de tamaño N. En tales casos, los problemas de tamaño práctico usualmente no pueden ser resueltos exactamente. Resultados recientes en el campo de la complejidad computacional, sugieren fuertemente (pero todavía sin probarlo) que los recursos computacionales requeridos para encontrar soluciones exactas de muchos de esos problemas combinatorios no pueden ser limitados por un polinomio en el tamaño de la cadena de los datos de entrada. Tales problemas son los NP-completos.

En muchas situaciones prácticas, una solución exacta no es necesaria. Cualquier solución cuyo costo está dentro de un rango del costo óptimo es aceptada.

3.1 La técnica de ramificación y acotación en los problemas de itineración.

Esta técnica ha sido muy estudiada por diversos investigadores para aplicarla en la resolución de los problemas de itineración. En el caso del problema "flow shop" pocos resultados existen para alguna medida de ejecución diferente al máximo tiempo de terminación (makespan). El trabajo principal fue desarrollado por Ignall y Schrage [19] e independientemente por Lomnicki [25].

En el problema "job shop" la efectividad de cualquier técnica dependerá del cálculo de buenas cotas inferiores para itinerarios parciales. Se ha establecido que para la ramificación y acotación usada en la itineración la eficiencia de una cota particular, frecuentemente dependerá de los datos del problema dado y también de la medida de ejecución empleada. En este problema también, la única medida de ejecución que ha recibido una atención importante en las investigaciones es el máximo tiempo de terminación. El trabajo fundamental de esta área fue realizado por Brooks y White [5]. Más recientemente, refinamientos de técnicas análogas han sido desarrolladas en el contexto del modelo más general que incluye itineración de recursos restringuidos.

Uno de los más significativos desarrollos de la ramificación y acotación en el problema "job shop" fue el reconocimiento del papel complementario que juegan los trabajos y las máquinas en el cálculo de la cota inferior.

El cálculo de la cota inferior basado en los trabajos, asume que no existen conflictos con los recursos, cuando el resto de los procesamientos de todos los trabajos es itinerado tan compactamente como sea posible. Este cálculo está basado en un itinerario parcial dado PS_t y un conjunto de operaciones asociadas S_t. En cualquier etapa, habrá una operación en el conjunto S_t para cada trabajo aún no completado. Para cada operación j en el conjunto S_t sea C_j el tiempo más temprano en el cual su procesamiento podría ser iniciado y sea R_j un procesamiento no itinerado para el trabajo correspondiente a la operación j. Entonces, para acomodar este trabajo en particular, la longitud del itinerario debe ser al menos $C_j + R_j$. Por lo tanto la forma de cota inferior basada en el trabajo será:

$$b_1 = \max_{j \in S_t} \{ C_j + R_j \}$$

Una cota complementaria basada en las máquinas asume que no existe un conflicto lógico (de precedencia) cuando el resto de los procesamientos para todas las máquinas es itinerado tan compactamente como sea posible. Este cálculo también está basado en un itinerario parcial dado PS_t, en donde la última terminación de una operación en una máquina k ocurre en el tiempo f_k. Sea M_k el procesamiento no itinerado que requerirá la máquina k. Entonces para acomodar la carga en la máquina k, la longitud del itinerario debe ser al menos $f_k + M_k$. Por lo tanto la forma de la cota inferior basada en la máquina será:

$$b_2 = \max_{1 \le k \le n} \{f_k + M_k\}$$

Y se obtiene la cota inferior final: $B = \max \{b_1, b_2\}$

Muchos autores que tratan con los resultados del desarrollo de un código de computadora para ramificación y acotación, no discuten la manera en que los desarrollaron, hay algunas buenas razones por lo cual no lo hacen:

a) La programación en computadora es una actividad individualista, y un programador rara vez está de acuerdo con otra organización y lógica.

b) La manera más conveniente de implantar una estrategia dada puede depender del lenguaje de programación usado.
c) Una estrategia dada puede ser obviada por las peculiaridades de la computadora usada.
d) Puede ser la única manera en que una estructura de un problema especial se ha desarrollado.

Sin embargo, existe un esquema adecuado de ramificación y acotación para los problemas de itineración, diseñado por [3]. En este esquema se utiliza la estrategia de "backtrack", debido a que con esa estrategia las demandas de almacenamiento en memoria no son muy severas, y en el caso de los programadores novatos, se evita hacer complicadas estructuras de datos para ahorrar memoria.

La "backtrack" es descrita como una estrategia de siempre ramificar del nodo que corresponde al más cercano subproblema resuelto (esto se conoce como el nodo actual en cada tiempo). De una manera más explícita, esto significa que la estrategia trata de encontrar una solución completa de la forma más directa, y entonces retroceder sus pasos sólo lo necesario para encontrar la siguiente solución más cercana en el árbol, esta estrategia se mencionó en el capítulo anterior como DF/LLB. La lógica elemental de esta estrategia se representa en el diagrama de la figura 3.1.

La descripción de cada paso del diagrama es la siguiente:

Inicio. Al comienzo todos los contadores, apuntadores y constantes deben ser inicializadas, y al menos es necesario generar el nodo inicial (que corresponde a una secuencia parcial nula) y colocarla en la lista de itinerarios parciales activos. También es una buena idea en este punto, construir una solución completa, por medio de un proceso heurístico rápido, ya que contar con una solución anticipada puede ser de valor para el algoritmo.

Ramificación. Se remueve el nodo actual del árbol y se reemplaza con los nodos formados por la asignación de la siguiente posición abierta en la secuencia. Es en este paso, cuando las propiedades de dominación deben ser verificadas, así esos nodos dominados son descartados, en lugar de ser puestos en el árbol. También en este paso, las cotas inferiores son calculadas para intentar sondear nodos y no ponerlos en el árbol.

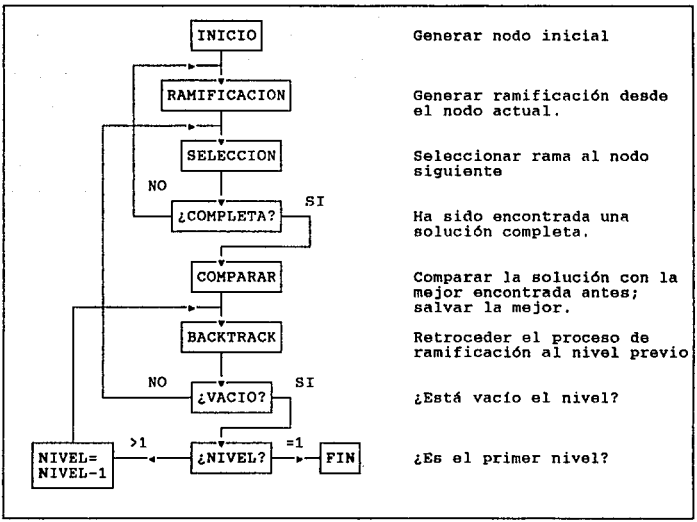


FIGURA 3.1. Esquema de r-a usando backtrack.

Selección. Entre los nodos más recientemente agregados al árbol, se selecciona uno con la mínima cota inferior, para ser el nuevo nodo actual. Se actualiza la información acumulativa, que debe ser almacenada para mantener una descripción suficiente del nodo actual.

Completa. Determina si el nodo actual es una solución completa. Si no se regresa a Ramificación, de otro modo se prosigue.

Comparar. Calcular el valor de la función objetivo asociado con la solución nueva encontrada, y compararla con la mejor solución encontrada que está en consideración. Considerar la mejor de éstas dos.

Backtrack. Retroceder el proceso de ramificación, retornando al nodo desde el cual el nodo actual fue creado. Actualizar la información almacenada, considerando este nodo, al sacar el trabajo en la última posición asignada en secuencia. Este nodo se convierte en el nodo actual.

Vacío. Determina si existen algunos nodos restantes, que fueron previamente generados desde el nodo actual. Si es así, regresar a Selección; de otro modo proseguir.

Nivel. Examinar si el nivel vacío acabado de encontrar, era realmente el primer nivel. Si es así, esto significa que todas las soluciones han sido implícitamente enumeradas y la solución que actualmente se considera es óptima. Si el nivel vacío no es el primero, entonces regresar a "backtrack".

3.2 El problema de itineración con restricciones de precedencia y perfil.

El objetivo de la itineración determinística, es obtener algoritmos eficientes, bajo el supuesto de que toda la información acerca de los trabajos y procesadores a ser itinerados es conocida por adelantado. Uno de los problemas fundamentales determinísticos, es el itinerar un conjunto de trabajos parcialmente ordenados, con tiempo de ejecución de uno, en un número de procesadores idénticos.

Cuando en una itineración, el conjunto de trabajos es dependiente, es decir, que cada trabajo depende de algún otro para su realización y el criterio de optimización impuesto es el de minimizar el máximo tiempo de terminación, la complejidad para la resolución de dicho problema es muy grande. Los resultados fundamentales de este problema están basados en el trabajo de Hu en el año de 1961 [16]. En esa investigación se trata con la itineración de trabajos con igual tiempo de ejecución en M procesadores idénticos, y con restricciones de precedencia. En la década de los setentas J.D. Ullman [31] demostró que una generalización del problema planteado por Hu es del tipo NP-completo. Para hacer esta afirmación se basó en la siguiente definición del problema, el que evidentemente demostró ser NP-

completo.

Definición: El problema de itineración con una unidad de tiempo de ejecución (UET), consiste en determinar si existe un itinerario para n trabajos en m procesadores, con $t_i = 1$ para $i = 1,\ldots,n$, con un límite de tiempo w, y restricción de precedencia $<$.

El problema de itineración UET está dado por una gráfica de precedencia y un perfil. La gráfica de precedencia, es una gráfica aciclica dirigida, que especifica la restricción de precedencia entre los trabajos UET. El perfil es una secuencia de números naturales que especifica la cantidad de procesadores disponibles en las etapas primera, segunda, tercera, etc. Cuando el perfil está formado por números iguales en todas las etapas, se dice que el perfil es recto. La amplitud del perfil es la cota superior en el número de procesadores que están disponibles en cualquier etapa del perfil.

Ullman demostró que decidir si existe un itinerario de un UET, para un perfil de longitud fija, es NP-completo, si la gráfica de precedencia es arbitraria, y el perfil no tiene amplitud y es recto. Lenkstra y Rinnooy Kan [23], probaron que el problema de decisión anterior permanece como NP-completo, aún si la altura de la gráfica de precedencia es de uno, y el perfil contiene tres etapas con un número arbitrario de disponibilidad en cada una de ellas. Dolev y Warmuth demostraron que de igual manera si se restringe el número de etapas con un número arbitrario de disponibilidad de uno, el problema es NP-completo.

Una variación del problema UET fue hecha por D. Dolev, E. Upfal y M. Warmuth [10], en el año de 1985, denominandolo como **"problema de itineración de trabajos de igual tiempo de ejecución con restricciones de precedencia y perfil de tiempo"**. El problema se encuentra definido de la siguiente manera:

Se cuenta con n trbajos con igual tiempo de ejecución y con retricciones de precedencia y perfil de tiempo. El objetivo es encontrar un itinerario el cual minimice el máximo tiempo de terminación.

Este problema lo denomina Ullman [7] como UET modificado, y ha demostrado que también es del tipo NP-completo.

Definición: El problema UET modificado consiste en determinar si existe un itinerario para n trabajos, con $t_i = 1$, para $i = 1,\ldots,n$, con un límite de tiempo w y restricción de precedencia $<$, y con

m(i) procesadores disponibles en el tiempo i para i= 1,...,w.

Los supuestos relacionados con este problema son los siguientes:

a) Todos los procesadores son idénticos y cualquier trabajo puede ser ejecutado en cualquiera de ellos.

b) Existe un orden parcial de los trabajos. Un trabajo no puede ser ejecutado si uno de sus trabajos antecesores (si existe) aún no ha sido ejecutado.

c) Cuando un procesador está ocioso y un trabajo está disponible, (i.e. que todos sus antecesores hayan sido procesados) éste debe comenzar a ejecutarlo.

d) Cada trabajo toma igual tiempo para su ejecución.

e) Un perfil de tiempo está dado, el cual especifica el número de procesadores que pueden ser ejecutados simultáneamente en cada etapa.

f) El objetivo de este problema es minimizar el máximo tiempo de terminación, el cual es la etapa en que el último trabajo es terminado.

El antecedente que se tiene en la resolución de este problema, es el trabajo de investigación realizado por D. Dolev y M. K. Warmuth en el año de 1984 [9]. Ellos propusieron un algoritmo basado en la programación dinámica, el cual tiene un tiempo de complejidad de:

$$O(n^{(h[m-1]+1)})$$

Donde: n es el número total de trabajos.
h es la altura de la gráfica de precedencia
m es la amplitud del perfil de tiempo.

Ellos probaron que si la amplitud del perfil está acotada por una constante "m", y si la altura de la gráfica de precedencia está acotada por una constante "h", entonces existe un algoritmo para encontrar la solución óptima. El número de procesadores disponibles en el perfil que ellos consideraron varía con el tiempo, esto es importante porque las anteriores aproximaciones recurrían a añadir componentes falsos a la gráfica para hacer que el perfil fuera recto.

En su trabajo primeramente se presenta una propiedad que algunos itinerarios óptimos tienen, y se muestra que solamente itinerarios polinomialmente de alto grado tienen esa propiedad. Debido a que su algoritmo utiliza programación dinámica para buscar todos los posibles itinerarios con esa propiedad, él encuentra un itinerario óptimo. El algoritmo usa recursión sobre la altura de la gráfica de precedencia, y se mantiene como polinomial cuando las gráficas tienen su altura acotada. Pero para algunas clases específicas de ellas, se puede utilizar la recursión sobre su estructura, obteniendo de manera similar algoritmos polinomiales.

La más reciente investigación en la resolución de este problema fue realizada por los doctores Cheng-I Yang, Jia-Shung Wang y R. C. T. Lee en el año de 1989 [32]. Ellos propusieron una técnica de ramificación y acotación para resolverlo, que básicamente consiste de cinco estrategias operativas (presentadas en el capítulo 2) para reducir el número de nodos recorridos en el árbol solución. Algunas de ellas son reglas heurísticas y otras son reglas de terminación de nodos.

Este algoritmo es diferente a algún otro usado en la resolución de problemas de itineración, debido a que en todos los demás el principal enfásis del proceso de solución, está dirigido a calcular las cotas inferiores, sin embargo para éste radica no sólo en hacer buenos cálculos de aquéllas, sino también al propósito de eliminar la generación de varios nodos del árbol solución, recortandolo al máximo, evitando cálculos innecesarios.

La relación que existe entre ambas técnicas usadas para la resolución del problema en cuestión, es que el espacio de búsqueda fundamental puede ser modelado como un árbol dirigido finito de soluciones parciales, y el objetivo es localizar y aislar una solución óptima enumerando explícitamente sólo una pequeña porción de la totalidad del árbol. Una conclusión dada por W.H. Kohler y K. Steiglitz [7] con respecto a la relación existente entre la programación dinámica y la técnica de ramificación y acotación, es que siempre es posible encontrar un algoritmo de ramificación y acotación equivalente a un algoritmo de programación dinámica que resuelve un problema de secuenciación.

Otro aspecto importante es que el algoritmo Yang, Wang y Lee resuelve el problema de forma más eficiente que el algoritmo de Dolev y Warmuth, y como se mostrará en el capítulo 4, su eficiencia se mantiene constante aún en problemas muy complicados.

3.3 Algoritmo de ramificación y acotación para resolver el problema de itineración con restricciones de precedencia y perfil.

En el algoritmo se utilizan primero las reglas 1, 2 y 5, debido a que son reglas de terminación de nodos, y eso permite que el árbol solución sea recortado. La aplicación secuencial de estas reglas, puede ser arbitraria. Las reglas 3, 4 y 5', son heurísticas, y se utilizan como criterio para encontrar la alternativa más efectiva. Debido a que son reglas heurísticas, pueden existir anomalías entre ellas, por ello la ejecución del algoritmo depende de la precedencia entre ellas. Una conclusión de los doctores, es que el orden de aplicación de ellas dependerá del problema a resolver. En el algoritmo propuesto, ellos presentan la siguiente secuenciación de las reglas: 3,4 y 5'.

Este algoritmo minimiza el máximo tiempo de terminación de un conjunto de trabajos. Ya que la longitud del itinerario depende del tiempo total de ejecución de los trabajos y del tiempo total en el que los procesadores permanecen ociosos. Al minimizar la longitud del itinerario se minimiza el tiempo de ocio, conduciendo a maximizar la utilización de los procesadores.

El algoritmo se presenta a continuación en forma de procedimiento recursivo, y la notación que se emplea es la siguiente:

P (S,i) =Itinerario parcialmente desarrollado en la etapa i
a.i. = Número acumulado de procesadores ociosos en la etapa i.

PROCEDIMIENTO SCHEDULE.

PASO 1. Conocer datos de entrada:

ACTUALES CANDIDATOS= Conjunto de trabajos en actual disponibilidad de ser ejecutados.
CAND= Conjunto de nodos internos.
ZCAND= Conjunto de nodos hoja.
m= Disponibilidad en el perfil actual.
s= Solución factible actual.

{Regla 2. Los nodos internos de la gráfica de precedencia se procesan antes que los nodos hoja}

PASO 2. SI CAND diferente del conjunto vacío ENTONCES

2.1 SI /CAND/ es mayor igual que m ENTONCES

2.1.1 Chequeo de sucesores comunes.
{Regla 1. Los nodos de la gráfica de precedencia que comparten sucesores comunes deben ser recorridos sólo una vez en el árbol de solución}

2.1.2 Generar todas las posibles alternativas.

2.2 EN CASO CONTRARIO,
Seleccionar aleatoriamente (m-/CAND/) nodos de ZCAND y todos los nodos de CAND.
FIN DEL CONDICIONAL 2.1.

2.3 MIENTRAS aún existan alternativas de 2.1.2 HACER
{Regla 3. Sean V_1 y V_2 dos nodos del árbol solución con el mismo predecesor inmediato, entonces si V_1 tiene un itinerario parcialmente desarrollado con cota inferior menor que la de V_2, implica que V_1 debe ser procesado antes que V_2}

2.3.1 Seleccionar los nodos del árbol solución con la mínima cota inferior.
{Regla 4. Un itinerario S no será solución óptima si P(S,i) está contenido en P(S',i) para algún S'}

2.3.2 SI hay más de un nodo con la mínima cota inferior, ENTONCES se selecciona uno que no esté limitado por P(S,i) de la solución factible
{Regla 5'. Sean V_1 y V_2 dos nodos del árbol solución con el mismo predecesor inmediato, si el conjunto de candidatos de V_1 es mayor que el de V_2, entonces V_1 debe ser procesado antes que V_2}

2.3.3 SI todavía hay problema ENTONCES
seleccionar uno de los nodos que tenga el máximo tamaño de su conjunto ACTUALES CANDIDATOS en la siguiente etapa de itineración.

FIN DEL CONDICIONAL 2.3.3.

FIN DEL CONDICIONAL 2.3.2.
{ Regla 5. Cualquier itinerario parcialmente desarrollado con a.i. mayor que un itinerario factible no será mejor que la solución factible, entonces será terminado}

```
   2.3.4  SI  no se tiene una solución factible  ENTONCES
          calcular:  P(S,i), cota superior, a.i
          LLAMAR A SCHEDULE.

   FIN DEL CONDICIONAL 2.3.4.

   2.3.5  Siguiente alternativa.

 FIN DEL MIENTRAS  2.3.

PASO 3. EN CASO CONTRARIO

        Itinerar todas las hojas secuencialmente.

FIN DEL CONDICIONAL DEL PASO 2.

PASO 4. REGRESAR.
```

CAPITULO 4

RESULTADOS EXPERIMENTALES

En este capítulo, se presenta el diseño de programa empleado, y la resolución del problema de itineración de trabajos de igual tiempo de ejecución con restricciones de precedencia y perfil a través del algoritmo de ramificación y acotación, comparandose con la resolución generada a través del algoritmo de programación dinámica de Dolev y Warmuth.

4.1 Estructura general del programa.

El programa de computadora del algoritmo de los doctores Cheng, Wang y Lee, se presenta estructurado un poco diferente a la manera en que ellos lo presentaron (3.3), debido a que las restricciones de memoria para nuestro desarrollo son mayores. En él se utiliza la técnica de backtracking (mencionada en el capitulo anterior) obteniendo primeramente una solución factible, para utilizar las cotas que ésta genera (número de etapas utilizadas y número de procesadores ociosos), en la eliminación de nodos, mediante una constante comparación de los mismos durante cada selección. El diagrama de flujo del programa, se encuentra en la figura 4.1 en forma de procedimiento recursivo. El programa se encuentra codificado en el lenguaje Pascal, y está conformado por varios módulos, en donde su interrelación es un poco complicada, debido a que algunos de éstos contienen procedimientos y funciones recursivas, y en los módulos clave la recursividad se hace presente en numerosas ocasiones al llamar a modulos recursivos en forma recursiva o en forma iterativa. Estas formas recursivas se obtuvieron a través del estudio de los casos codificados en forma iterativa, analizando casos con dos ciclos iterativos, con tres ciclos , con cuatro, ..., hasta que se obtenía la generalización para varios, lo que permitía, al reconocer las variables y constantes, establecer los parámetros del procedimiento recursivo. Esto permitió que el programa no tuviera una gran cantidad de líneas y que fuera más eficiente (con respecto a la memoria usada y velocidad), sin embargo, complicó el entendimiento de su funcionamiento. Con respecto a los nombres empleados, la mayoría de los procedimientos, funciones, variables, constantes, parámetros y tipos, lo tienen muy fácilmente identificable con su uso. La codificación se presenta en el apéndice A.

Las estructuras de datos más importantes que se emplean en el programa, son las siguientes:

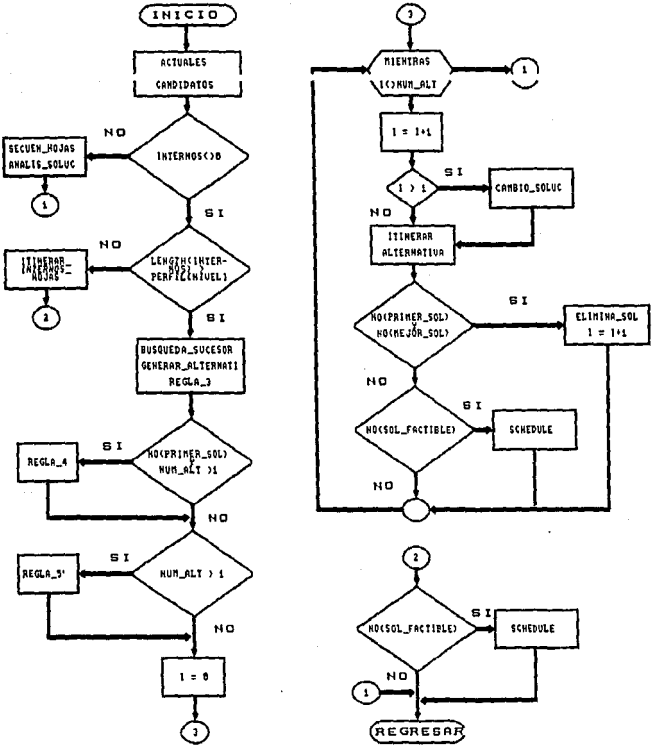


Figura 4.1 Diagrama de flujo del programa.

Una matriz booleana, que representa la adyacencia de los nodos de la gráfica de precedencia.

Un arreglo de registros, que representa las soluciones obtenidas, cada registro cuenta con tres campos , el primero que contiene la itineración de los trabajos en un arreglo, el segundo que contiene la cota de la solución (regla 3) y el tercero que contiene el número de procesadores ociosos (regla 5). Así por ejemplo, si se tiene el siguiente itinerario parcial PS, con sus respectivos cota inferior y número de procesadores ociosos:

PS = {1,2,*,3,4,5} C.I. = 6 a.i = 1

La representación en la estructura de datos es:

```
Nodo_solución.Conjunto[1]        = 1
Nodo_solución.Conjunto[2]        = 2
Nodo_solución.Conjunto[3]        = -1
Nodo_solución.Conjunto[4]        = 3
Nodo_solución.Conjunto[5]        = 4
Nodo_solución.Conjunto[6]        = 5
Nodo_solución.Cota               = 6
Nodo_solución.Numero_de_ociosos  = 1
```

Nota: el -1 representa a un procesador ocioso.

Sin embargo como el programa encuentra otras soluciones, y es necesario manejarlas, esta estructura se tuvo que agrupar como un arreglo, así la estructura resultante es un arreglo de nodos solución.

Un arreglo de registros, que representa los sucesores de los trabajos, cada registro cuenta con 3 campos, el primero que contiene el sucesor_comun de un grupo de antecesores, representado por el segundo campo y el tercer campo contiene el número de antecesores. Así por ejemplo, si se considera la gráfica de precedencia y conjunto de nodos internos de la figura 4.2:

La Representación en la estructura es la siguiente:

```
Nodo_sucesor.Antecesor[1]        = 1
Nodo_sucesor.Antecesor[2]        = 2
Nodo_sucesor.Antecesor[3]        = 3
Nodo_sucesor.Sucesor_comun       = 4
Nodo_sucesor.Num_de_antecesores  = 3
```

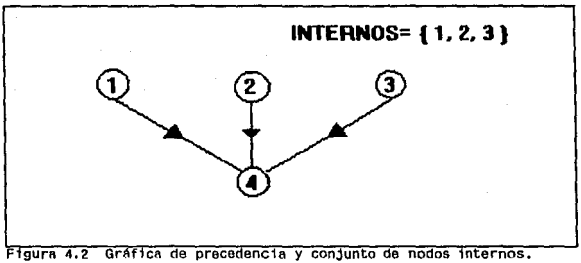


Figura 4.2 Gráfica de precedencia y conjunto de nodos internos.

De la misma forma que el caso anterior, era necesario manejar varios Nodo_sucesor, ya que pueden existir diferentes sucesores comunes en una gráfica (como se muestra en el siguiente inciso), por lo que esta estructura se agrupó en un arreglo.

Una matriz de enteros, en donde cada renglón representa las combinaciones factibles para itineración, de acuerdo a la regla 1 y la regla 2. Así por ejemplo, si se tienen las siguientes combinaciones de elementos:

(2,3,4,5,6), (2,3,5,6,7), (2,4,5,6,7), (3,4,5,6,7)

La representación en la estructura genera la siguiente matriz:

$$\text{Alternativa} = \begin{bmatrix} 2 & 3 & 4 & 5 & 6 \\ 2 & 3 & 5 & 6 & 7 \\ 2 & 4 & 5 & 6 & 7 \\ 3 & 4 & 5 & 6 & 7 \end{bmatrix}$$

Con lo que respecta a las entradas y salidas del programa, se tiene que el máximo número de trabajos que pueden ser procesados por el programa es 50, y el tamaño máximo de una etapa del perfil de tiempo es de 7 procesadores, con una longitud máxima de 14.

La gráfica de precedencia no tiene ningún orden en la manera en que se encuentran los nodos, esto significa que puede ser que el nodo con la máxima prioridad o jerarquía sea cualquiera diferente del número 1, y que el número 1 puede estar en cualquier nivel. Más aún, es posible que a cada nodo se le asigne un nombre, con el objeto de hacer más útil el programa.

Una vez que se ha introducido la matriz de adyacencia, se presenta en pantalla la gráfica de precedencia correspondiente, para la verificación de los datos y poderlos cambiar en caso necesario. Al finalizar los cálculos se presenta la solución óptima encontrada, así como el recorrido óptimo de dicha solución dentro de la gráfica de precedencia.

El programa presenta un desplegado y la solución parcial obtenida en el caso de que el perfil de tiempo dado, no hubiera sido lo suficientemente grande para realizar una itineración total de los nodos.

Dentro de la estructura general del programa, se puede mencionar que los procedimientos que forman la parte principal son: Actuales_candidatos, Búsqueda_de_sucesores_comunes, Generar_alternativas, Regla_3, Regla_4 y Regla_5'.

4.2 Procedimientos y funciones principales.

Procedimiento Actuales Candidatos.

Este procedimiento se utiliza para identificar todos los nodos sucesores de una solución parcial y clasificarlos como conjuntos de nodos hoja o de nodos internos. Por ejemplo, si se considera la gráfica de precedencia y solución parcial de la figura 4.3, al aplicar este procedimiento se generan los conjuntos ahí mostrados.

Este procedimiento emplea para su funcionamiento el campo "conjunto", del i-ésimo registro Nodo_solución. El análisis de cada elemento del conjunto mediante la matriz de adyacencia, i.e. inspección sobre el renglón "conjunto[k]", determina si en la j-ésima columna existe un valor TRUE (que significa la adyacencia al nodo j), y si ese j-ésimo elemento no pertenece a Actuales y además todos sus nodos predecesores están itinerados (se emplea la función Elegible), entonces se le asigna como miembro de Actuales. Posteriormente se analiza si dicho elemento tiene sucesores, lo que

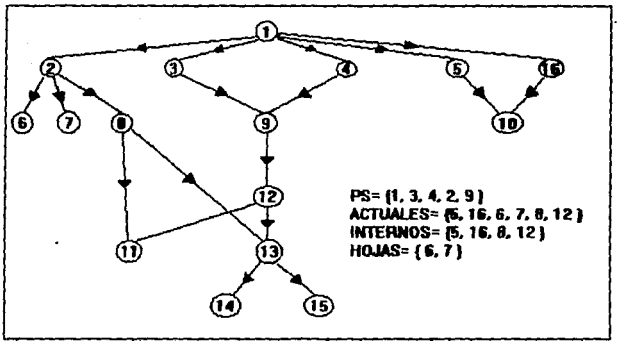


Figura 4.3 Gráfica de precedencia y solución parcial (PS).

determina si se le clasifica como miembro del conjunto Internos o del conjunto Hojas.

Procedimiento Búsqueda_de_sucesores_comunes.

Este procedimiento se utiliza para identificar todos los sucesores comunes del conjunto internos. Sólo se analiza el conjunto internos debido a que "la Regla 2" especifica que los nodos internos se deben de itinerar antes. En este análisis, se identifican grupos de nodos que son antecesores comunes de algún nodo, cada uno de estos grupos es sumamente importante, ya que constituyen la base para generar las combinaciones factibles de nodos (llamadas alternativas), que se utilizan en cada itineración.

Debido al mecanismo de funcionamiento del procedimiento Generar_alternativas, los grupos que aquí se generan deben de ser analizados inmediatamente después de ser creados.

Un ejemplo del funcionamiento, es el que a continuación se menciona, si consideramos la gráfica de precedencia y el conjunto de internos de la figura 4.4:

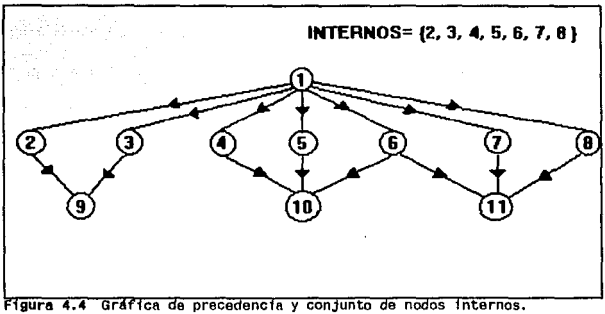


Figura 4.4 Gráfica de precedencia y conjunto de nodos internos.

El procedimiento encuentra los siguientes grupos:

Sucesor Común	Antecesores
9	2,3
10	4,5,6
11	6,7,8

Sin embargo, el análisis se complica cuando se considera el siguiente ejemplo de la figura 4.5:

Se tienen los siguientes grupos:

Sucesor Común	Antecesores
8	2
9	2
10	2
11	3,4,5
12	3,4,5,6
13	6
14	6,7

En donde el análisis posterior a la generación de los mismos

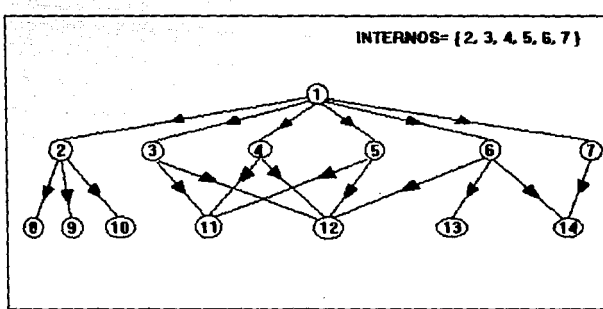


Figura 4.5 Gráfica de precedencia y conjunto de nodos internos.

produce únicamente los siguientes grupos:

Sucesor Común	Antecesores
8	2
12	3,4,5,6
14	6,7

Esta simplificación de grupos es vital, ya que en la generación de alternativas, se toma uno o varios elementos de cada grupo en conjunción con los de otro grupo, formando así una combinación factible, que cumple con lo estipulado en el teorema 1.

De no hacer esto se tendría, por ejemplo para una combinación de dos elementos (donde el primero pertenece a un grupo y el segundo a otro), con los primeros tres sucesores comunes {8,9,10}: (2,2) , (2,2) que es completamente absurdo, por lo tanto, sólo se considera al grupo cuyo sucesor común es el 8.

Y de la misma manera si se consideran los grupos de los sucesores comunes 11 y 12, se observa que al hacer una combinación de dos elementos (el primer elemento pertenece al grupo 11 y el segundo al grupo de 12), se tendría por ejemplo: (3,3), (3,4), (3,5), en donde sólo (3,4) o (3,5), tienen sentido (no se pueden

considerar ambos al mismo tiempo, debido a la regla 1). Por lo tanto cuando un grupo sea igual o este contenido en algún otro, debe de ser eliminado.

Este procedimiento emplea para su funcionamiento el k-ésimo elemento del conjunto internos, y lo analiza mediante la matriz de adyacencia, buscando sobre el renglón "internos[k]", si se encuentra un valor "TRUE" en la columna j, y ese j-ésimo elemento no ha sido analizado como sucesor común (se emplea la función Ya_analizado), entonces se analiza dicha columna, en el momento que se tenga un valor "TRUE" del j1-ésimo renglón, se considera si j1 pertenece al conjunto internos, si pertenece se hace la siguiente asignación:

```
sucesor_comun = j
antecesor[i]  = j1
```

Posteriormente se hace el análisis, ya mencionado de los grupos, mediante el procedimiento Simplificación_de_sucesores_comunes.

Procedimiento Generar_alternativas.

Este procedimiento se utiliza para hacer las combinaciones factibles entre los grupos de sucesores comunes, de acuerdo con el teorema 1 (capítulo 2). Debido a lo complejo de la selección de los elementos de cada grupo, este procedimiento es el más complicado de todos, por las razones que a continuación se exponen.

Considerese por ejemplo, el caso más sencillo de una combinación, la que únicamente involucra dos elementos, considerese la gráfica de precedencia y el conjunto de internos de la figura 4.6:

Las posibles combinaciones que se obtienen son:
(2,3), (2,4), (2,5), (2,6), (3,4), (3,5), (3,6), (4,5), (4,6)
pero debido al teorema 1, las combinaciones factibles son:
(2,3), (2,4), (2,6), (4,5), (4,6)

*Ya que por ejemplo: (2,4) es equivalente a (3,4), puesto que 2 y 3 tienen el mismo sucesor comun, y el segundo elemento de la combinación es el mismo.

Este mecanismo de generación, puede ser descrito de la manera siguiente:

a) Hacer una combinación, si es posible, de cada grupo de antecesores comunes. Generando: (2,3) y (4,5).

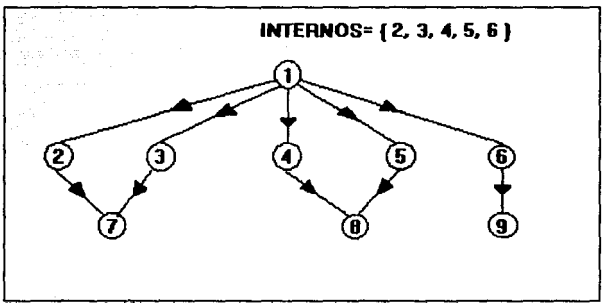


Figura 4.6 Gráfica de precedencia y conjunto de nodos internos.

b) Hacer una combinación, tomando un elemento de cada grupo. Generando: (2,4), (2,6) y (4,6).

Sin embargo, si se considera por ejemplo la figura 4.7:

El mecanismo anterior no sirve, ya que los grupos que genera el procedimiento Búsqueda_de_sucesores_comunes son:

Sucesor Común	Antecesores
7	2,3
8	3,4
9	4,5
10	5,6

Y el empleo de dicho mecanismo generaría (2,3), (3,4), que son iguales al considerar el teorema 1. Esto se debe, al hecho de que generar combinaciones factibles con grupos de antecesores comunes independientes (i.e. que ningún elemento pertenece a dos grupos al mismo tiempo), es relativamente fácil.

Es por ello, que el mecanismo de generación debe ser más complicado, de tal manera, que genere, las siguientes combinaciones factibles, que visiblemente se pueden generar fácilmente, pero

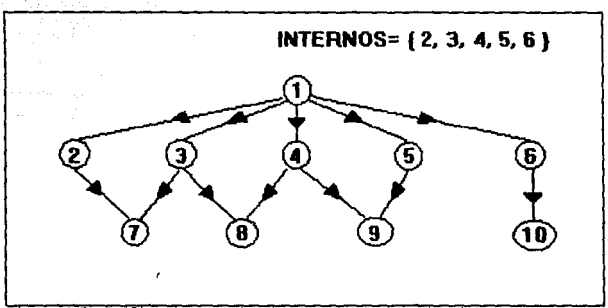


Figura 4.7 Gráfica de precedencia y conjunto de nodos internos.

computacionalmente resulta un poco complicado: (2,3), (2,5), (2,6), (4,5), (4,6).

Y más aún, si se considera por ejemplo la figura 4.8, donde la búsqueda de sucesores comunes produce:

Sucesor Común	Antecesores
8	2,3,4,5
9	4,5,6
10	6,7

La generación de combinaciones factibles, visiblemente es complicada, y produce: (2,3), (2,6) y (6,7).

Es por ello, que este procedimiento es el más complejo de todos, ya que como se ha mostrado, existen casos muy complicados en los que el encontrar las combinaciones factibles de dos elementos resulta muy difícil. Se debe pensar, en el caso en que se requieren combinaciones de más de dos elementos, evidentemente, se complica aún más la búsqueda, sin embargo un extenuante estudio del comportamiento de las combinaciones, hizo finalmente que se pudiera elaborar el procedimiento computacional, capáz de encontrar todo tipo de combinaciones, por más complicada que sea la gráfica de

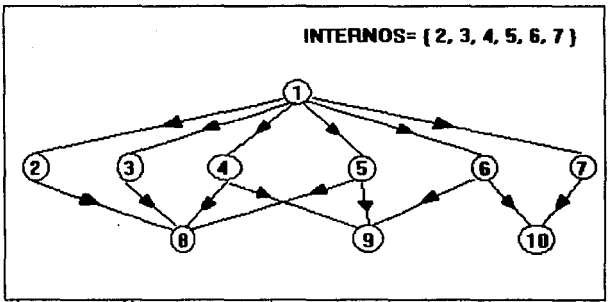


Figura 4.8 Gráfica de precedencia y conjunto de nodos internos.

precedencia.

Procedimiento Regla_3.

Este procedimiento, calcula la cota inferior de un conjunto de alternativas (combinaciones factibles), y selecciona a aquellas alternativas que tengan la mínima cota inferior.

El procedimiento utiliza 4 procedimientos adicionales (Jobs_itinerados, Jobs_sin_itinerar, Cota_conjunto, Mínima_cota) y una función (Perfil_acumulado) durante su funcionamiento. La forma de operación es la siguiente:

A partir de un conjunto de alternativas, se comienza por calcular las soluciones parciales de cada una de ellas, por ejemplo si se tiene la siguiente solución parcial y las siguientes alternativas:

Solución parcial: {1,2,*,4,5,6}
Alternativas: (3,8), (3,9), (3,10).

Las soluciones parciales que se obtienen son respectivamente:

{1,2,*,4,5,6,3,8}, {1,2,*,4,5,6,3,9}, {1,2,*,4,5,6,3,10}

Entonces con estas soluciones parciales, se calcula el conjunto de trabajos sin itinerar (SUP), en donde se busca el trabajo que tenga la máxima altura dentro de la gráfica de precedencia, posteriormente se calcula la cota inferior de cada alternativa de acuerdo a la definición dada en el capítulo 2, es decir:

Cota inferior = Nivel del nodo solución + Máxima altura de SUP

Y se concluye, seleccionando a las alternativas con la menor cota inferior.

Procedimiento Regla_4.

Este procedimiento, se utiliza para comparar una solución parcial de un itinerario factible, con otra de una alternativa que se encuentra analizandose. Es importante mencionar que ésta regla, sólo se emplea cuando se tiene una solución factible que se considera óptima, y por ello durante la ramificación y acotación para encontrar la primera solución no puede ser utilizada.

El procedimiento utiliza para su funcionamiento dos procedimientos adicionales (Jobs_itinerados y Ordena) y una función (Examina). La forma de operación es la siguiente:

A partir de un conjunto de alternativas, se calcula las soluciones parciales de cada una de ellas, y se obtienen los conjuntos de trabajos itinerados, cada uno de ellos es comparado con el conjunto de trabajos itinerados de una solución que se considera óptima, para determinar si alguno está contenido en el conjunto óptimo, el procedimiento finaliza al seleccionar las alternativas que generaron conjuntos no contenidos en el conjunto óptimo.

Procedimiento Regla_5'.

Este procedimiento se utiliza para calcular el número posible de nodos que estarían en disponibilidad para ser itinerados, si se itinerara la alternativa(i).

El procedimiento emplea un procedimiento adicional (Actuales_candidatos) y una función (Max_cardinalidad) para su funcionamiento. La manera de operación es la siguiente:

A partir de un conjunto de alternativas, se itinera cada una de ellas y con la solución parcial obtenida se calcula el número de nodos en disponibilidad para ser itinerados, se concluye al

seleccionar las alternativas que generen el mayor número de nodos en disponibilidad para ser itinerados.

4.3 Resultados experimentales.

Los resultados experimentales que se presentan, se obtuvieron al hacer una comparación con el Algoritmo de Programación dinámica sugerido por Dolev y Warmuth, el cual tiene un tiempo de complejidad de $O(n^{(h(m-1)+1)})$, donde:

n es el número total de trabajos.
h es la altura de la gráfica de precedencia.
m es la amplitud del perfil de tiempo.

Los experimentos se obtuvieron a partir de las siguientes entradas:

Dado un número n de trabajos, una matriz de adyacencia M (de estos n trabajos) es generada aleatoriamente, usando un generador de numeros aleatorios Fortran de una computadora VAX 11/750. M representa la gráfica de precedencia, y M(i,j)=1 denota que existe un arco directo del nodo i al nodo j. Para satisfacer la restricción de no tener ciclos, M(i,j)=1 sólo para i>j. El perfil de tiempo está construído usando el mismo generador de números aleatorios.

Se probó el Algoritmo de ramificación y acotación con cinco conjuntos de datos: n=14, n=20, n=40, n=50, y n=100. Para n=20, se generaron 10 casos de prueba. Para n=40 y n=50, se generaron 4 casos, respectivamente. Y para n=50, 10 casos fueron generados. También se probó el Algoritmo de Dolev y Warmuth con el mismo número de casos de prueba. El programa fue codificado en PASCAL y corrido en una computadora VAX 11/750, bajo el sistema operativo VMS.

Para ajustar la esparción de la matriz de adyacencia M, se adoptó un parámetro "s", con $0.1 \leq s \leq 0.9$. La matriz esparcida M, fue construída mediante el siguiente procedimiento:

Inicialmente, el generador de números aleatorios elabora cada M(i,j). Si el valor de M(i,j) es más pequeño que "s", entonces se hace M(i,j)=0; de otro modo se hace M(i,j)=1.

Los resultados de la simulación, muestran que si el número de

trabajos es mayor que 30 o el parámetro de esparción es más pequeño que 0.5, el Algoritmo de Dolev y Warmuth sólo produce resultados después de haber corrido durante una semana. Mientras que, el Algoritmo de ramificación y acotación no tuvo ningún problema para producir resultados en ambos casos.

En la tabla 1 se muestran 16 casos para n=14, y el parámetro de esparción con un rango de 0.1 a 0.9 como se muestra en la primera columna. En la segunda y tercera columna se muestran, el número de nodos recorridos en el árbol solución generado por el Algoritmo de ramificación y acotación y el Algoritmo de Dolev y Warmuth, respectivamente. En estas dos columnas, hay que notar que el Algoritmo de Dolev y Warmuth se desempeña mejor en una matriz esparcida, debido a que la mayoría de los trabajos son nodos hoja de la gráfica de precedencia. Como para una matriz densa, el número de nodos recorridos en el árbol solución crece exponencialmente, el Algoritmo de Dolev y Warmuth va perdiendo eficiencia, sin embargo el Algoritmo de ramifiación y acotación se desarrolla bien para todos los casos generados. El tiempo de computadora (tiempo de CPU en segundos) de ambos se presenta en la quinta y sexta columnas. En la séptima columna, se presenta la razón entre el número de nodos del árbol solución recorridos por ambos.

En la tabla 2, se presentan los resultados experimentales para los casos n=20. El parámetro de esparción tiene un rango de 0.9 a 0.5, debido a que el Algoritmo de Dolev y Warmuth toma muchísimo tiempo para la corrida con $s < 0.5$, sólo se consideran los casos para $s \geq 0.5$. En esta tabla, se puede observar que el desempeño del Algoritmo de Dolev y Warmuth, se convierte en ineficiente con respecto al Algoritmo de ramificación y acotación conforme la matriz se vuelve densa.

En la tabla 3, sólo se presentan resultados experimentales para el Algoritmo de ramificación y acotación de los casos n=40, n=50, n=100. La primera columna muestra el parámetro de esparción. Cada dato de la segunda columna, muestra el número de nodos recorridos por el Algoritmo para n=40. La tercera columna, muestra el tiempo de computadora (tiempo de CPU en segundos). Los casos para n=50 y n=100 se encuentran listados en las siguientes cuatro columnas. Esta tabla muestra que el Algoritmo de ramificación y acotación se desempeña bien aún para casos grandes, mientras que el Algoritmo de Dolev y Warmuth no producen algún resultado eficiente para estos casos.

Paráme tro"s"	Número de nodos recorridos en el árbol solución.			Tiempo de CPU.(mlseg)		
	Y, W & L	D & W	Razón %	Y, W & L	D & W	Razón %
0.9	4	448	0.89	2.19	6.29	34.8
0.9	10	578	1.7	3.29	5.86	56.1
0.9	10	4644	0.21	3.15	23.8	13.0
0.8	9	9335	0.09	1.68	37.3	4.5
0.8	9	9355	0.09	1.64	37.3	4.3
0.8	20	9607	0.2	1.7	24.3	6.9
0.7	68	9615	0.7	1.95	18.3	10.6
0.7	48	9096	0.5	1.94	23.3	8.3
0.5	37	9335	0.39	1.86	37.3	4.9
0.5	32	67675	0.04	3.22	247.1	1.3
0.4	10	48814	0.02	3.15	102.1	3.0
0.3	16	56287	0.02	3.32	134.4	2.4
0.83	15	530112	0.002	3.14	1801.2	0.1
0.2	19	234310	0.008	3.20	816.7	0.3
0.2	7	204304	0.003	3.21	753.0	0.4
0.1	12	1458839	0.0008	3.16	4907.2	0.06

TABLA 1. Comparación para 14 trabajos itinerados.

Paráme tro"s"	Número de nodos recorridos en el árbol solución.			Tiempo de CPU.(mlseg)		
	Y, W & L	D & W	Razón %	Y, W & L	D & W	Razón %
0.9	25	448	5.5	2.53	6.29	40.2
0.8	43	2244	1.9	3.27	25.07	13.0
0.75	32	53372	0.05	3.41	283.66	1.2
0.75	32	32094	0.09	3.22	166.94	1.92
0.75	24	33197	0.07	3.33	169.3	1.96
0.75	37	32094	0.1	3.33	167.14	1.9
0.5	20	163256	0.01	3.42	748.12	0.4
0.5	20	172145	0.01	3.30	748.64	0.4
0.5	37	246706	0.01	3.52	831.85	0.4
0.5	48	301767	0.01	3.46	1141.94	0.3

TABLA 2. Comparación para 20 trabajos itinerados.

El hecho de que el Algoritmo de ramificación y acotación se desempeñe mejor que el Algoritmo de Dolev y Warmuth es debido a que el segundo está basado en la técnica de programación dinámica, que es esencialmente una aproximación de búsqueda exhaustiva. El

Algoritmo de ramificación y acotación puede recortar el espacio por las reglas de terminación y provee una búsqueda heurística en ese espacio disminuído.

Paráme tro"s"	Número de nodos recorridos en el árbol solución.			Tiempo de CPU.(mlseg)		
	40 TRAB.	50 TRAB.	100 TRAB	40 TRAB.	50 TRAB.	100 TRAB
0.9	77	290	1352	4.35	7.2	44.42
0.8	-	-	1139	-	-	36.75
0.7	31	279	95	4.39	6.7	13.17
0.6	-	-	76	-	-	8.34
0.5	15	258	21	3.28	6.58	8.03
0.4	-	-	69	-	-	8.86
0.3	-	-	116	-	-	18.36
0.2	-	-	89	-	-	13.63
0.1	20	169	74	3.85	5.5	8.46

TABLA 3. Resultados para 40, 50 y 100 trabajos itinerados.

CONCLUSIONES.

La teoría de itineración, cuenta con una ayuda muy importante de la técnica de ramificación y acotación, para resolver varios modelos que hasta el momento han recibido poca atención. Debido a que los avances de la computación en cuanto a memoria y velocidad de procesamiento, han permitido el desarrollo de la técnica.

Al considerar un diseño de algoritmo de ramificación y acotación para estos modelos, es importante tratar de disminuir el número de nodos generados en el árbol de solución, a través de alguna estrategia, la cual puede ser dictada por alguna de las cinco reglas presentadas, si el problema tiene cierta relación con el presentado aquí. Por ejemplo, los problemas de itineración sin restricciones de árbol discutidos en [6], el de itineración con restricciones de precedencia con cota de altura discutido en [22], el de itineración con tiempos de comienzo y vencimiento discutido en [27]. Cabe señalar que todos estos problemas son del tipo NP-completos.

El Algoritmo de Yang, Wang y Lee, por estar diseñado en forma de procedimiento recursivo, utiliza gran cantidad de memoria para problemas de tamaño mediano y grande. Sin embargo, se comprobó que en comparación con el algoritmo de Dolev y Warmuth, su desempeño es mejor y se mantiene constante aún en problemas grandes, mientras que el algoritmo de progamación dinámica, obtiene resultados en un tiempo cada vez mayor conforme el problema se vuelve más grande con respecto al número de nodos.

Para un problema en donde la gráfica de precedencia tiene una forma compleja, es decir en donde un nodo cualquiera cuenta con más de un predecesor inmediato, y además se tiene un perfil de tiempo mayor de uno, la generación de los posibles itinerarios es complicada, y su resolución depende directamente del procedimiento que genera las combinaciones factibles en cada etapa de itineración.

APENDICE A

```
{ ALGORITMO DE RAMIFICACION Y ACOTACION PARA RESOLVER EL PROBLEMA DE}
{ITINERACION DE TRABAJOS CON IGUAL TIEMPO DE EJECUCION Y RESTRICCIONES}
                {DE PRECEDENCIA Y PERFIL DE TIEMPO}

{$M 49152,0,655360}

PROGRAM ITINERACION;

Uses      crt, graph, dos;

Const     maxnodos              =50;      max_rango_tiempo      =14;
          max_rango_conj        =50;      max_rango_suc         =20;
          max_rango_bool        =50;      max_rango_alt         =50;
          maxtiempo             =7;       max_rango_conj_sol    =98;
          max_rango_ind1        =7;       maxsoluciones         =20;
          max_rango_ind2        =7;       max_rango_renglon     =98;

Type      nodptr                =-1..maxnodos;
          rango_conj            =-1..max_rango_conj;
          periodo               =0..maxtiempo;
          rango_tiempo          =0..max_rango_tiempo;
          rango_alt             =0..max_rango_alt;
          rango_bool            =0..max_rango_bool;
          rango_ind1            =0..max_rango_ind1;
          rango_ind2            =0..max_rango_ind2;
          rango_renglon         =0..max_rango_renglon;
          rango_suc             =0..max_rango_suc;
          rango_conj_sol        =0..max_rango_conj_sol;
          rango_sol             =0..maxsoluciones;

          matriz_nodos          =array[nodptr,nodptr] of boolean;
          conjuntos             =array[rango_conj] of nodptr;
          conjunto_sol          =array[rango_conj_sol] of nodptr;
          perfil_tiempo         =array[rango_tiempo] of periodo;
          alternativas          =array[rango_alt,periodo] of nodptr;
          arr_bool              =array[rango_bool] of boolean;
          arreglo_indices1      =array[rango_ind1] of rango_suc;
          arreglo_indices2      =array[rango_ind2] of rango_conj;
          renglones             =array[rango_renglon]  of integer;
          nombres               =array[nodptr] of string[10];
          respuestas            =string[2];

          nodo_solucion    = record
                               conjunto        :conjunto_sol;
                               cota            :rango_tiempo;
                               num_de_ociosos  :rango_conj_sol;
                             end;
          soluciones=array[rango_sol] of nodo_solucion;


          nodo_sucesor     = record
                               sucesor_comun   :nodptr;
```

```
                                num_de_ant      :nodptr;
                                antecesor       :conjuntos;
                              end;
          sucesores=array[rango_suc] of nodo_sucesor;


Var       matriz1                                       :matriz_nodos;
          solucion                                      :soluciones;
          sucesor                                       :sucesores;
          altura,actuales,internos,hojas                :conjuntos;
          perfil                                        :perfil_tiempo;
          nombre                                        :nombres;
          num_nodos                                     :nodptr;
          num_etapas                                    :rango_tiempo;

{********************** INICIALIZACIONES ***************************}

procedure inicia_matriz (var mat  :matriz_nodos);
var     i,j    :nodptr;
 begin
  for i:=-1 to maxnodos do
   for j:=-1 to maxnodos do
     mat[i,j]:= false;
 end;  {procedure}

procedure inicia_soluciones;
var i   :rango_sol;
    j   :rango_conj_sol;
begin
 for i:=0 to maxsoluciones do
  with solucion[i] do
   begin
    for j:=0 to max_rango_conj do conjunto[j]:=0;
    cota:=0;
    num_de_ociosos:=0;
   end; {with}
end;  {procedure}

procedure inicia_sucesor;
var  i :rango_suc;
     j :rango_conj;
begin
 for i:=0 to max_rango_suc do
  begin
   with sucesor[i] do
    begin
     sucesor_comun:=0;
     for j:=-1 to max_rango_conj do antecesor[j]:=0;
    end; {with}
  end;  {for i}
end; {procedure}

procedure inic1(var vector:conjuntos);
var i :rango_conj;
 begin
  for i:=-1 to max_rango_conj do
```

```
   vector[i]:=0;
 end;

procedure inic2(var vector:perfil_tiempo);
var i :rango_tiempo;
 begin
  for i:=0 to max_rango_tiempo do
   vector[i]:=0;
 end;

procedure inic3 (var x :alternativas);
var  i :rango_alt;
     j :periodo;
begin
 for i:=0 to max_rango_alt do
  for j:=0 to maxtiempo do   x[i,j]:=0;
end; {procedure}

procedure inic4 (var x:renglones);
var i  :rango_renglon;
begin
 for i:=0 to max_rango_renglon do x[i]:=0;
end; {procedure}

procedure inic5(var x:arreglo_indices1);
var i :rango_ind1;
begin
 for i:=0 to max_rango_ind1 do x[i]:=0;
end;  {procedure}

procedure inic6(var x:arreglo_indices2);
var i :rango_ind2;
begin
 for i:=0 to max_rango_ind2 do x[i]:=0;
end;  {procedure}

[***************** ITINERACION INICIAL *************************]

function perfil_acumulado (nivel :rango_tiempo) :rango_conj_sol;
var  s     :rango_conj_sol;
     k     :rango_tiempo;
begin
 s:=0;
 for k:=1 to nivel do s:= s + perfil[k];
 perfil_acumulado:=s;
end; {function}

procedure primera_itineracion
(var num_de_elementos:rango_conj_sol; var nivel:rango_tiempo);
var big        :nodptr;
    i,k,acum   :rango_conj_sol;
begin
 inicia_soluciones; k:=0; big:=altura[1];
 for i:=2 to num_nodos do  if altura[i]>big then big:=altura[i];
 with solucion[1] do
 begin
```

```
  i:=0;
  repeat
   i:=i+1;
   if altura[i]=big then
    begin
     k:=k+1;
     conjunto[k]:=i;
    end; {if}
  until (i=num_nodos);
  acum:=perfil[1];  nivel:=1;
  while (k > acum) do                    {actualizacion del nivel}
   begin
    nivel:=nivel+1;
    acum:=perfil_acumulado(nivel);
   end;
  for i:=k+1 to acum do conjunto[i]:=-1;      {procesadores ociosos}
  num_de_ociosos:=acum-k;
  cota:=nivel;
 end; {with}
 num_de_elementos:=k;
end; {procedure}

{***************** CALCULO DE ALTURA DE LOS NODOS *********************}

{funcion que determina si el calculo de la altura ha sido finalizado}

function finalizacion                              :boolean;
 var revision   :boolean;
     i          :rango_conj;
 begin
  revision:=true;
  i:=1;
   repeat
    if altura[i]=0 then revision:=false;
    i:=i+1;
   until (i>num_nodos) or not(revision);
   finalizacion:=revision
 end; {function}

procedure asignacion  (var matA,matB    :matriz_nodos);
var  i,j      :nodptr;
begin
 for i:=1 to num_nodos do
  for j:=1 to num_nodos do
   matB[i,j] :=matA[i,j];
end; {procedure}

procedure potencia (exponente :nodptr;var matriz3:matriz_nodos);
 var    i,j,k       :nodptr;
        matriz2     :matriz_nodos;
 begin
  if exponente=1 then asignacion(matriz1,matriz2)
          else asignacion(matriz3,matriz2);
  for i:=1 to num_nodos do
   for j:=1 to num_nodos do
    begin
```

```
      matriz3[i,j]:=false;
      for k:=1 to num_nodos do
        matriz3[i,j]:=matriz3[i,j] or (matriz1[i,k] and matriz2[k,j]);
    end;    {for j}
 end; {procedure}

{procedimiento que checa si hay una linea falsa en la matriz de adyacencia
potenciada para calcular la altura}

procedure chequeo   (matriz:matriz_nodos; exponente:nodptr);
 var busqueda     :boolean;
     i,j          :nodptr;
 begin
  for i:=1 to num_nodos do
   begin
    busqueda:=true;
    j:=1;
    repeat
     if matriz[i,j] then busqueda:=false;
     j:=j+1;
    until (j>num_nodos)or not(busqueda);
    if (busqueda) and (altura[i]=0) then altura[i]:=exponente+1;
   end;   {for}
 end;  {procedure}

procedure calculo_de_altura;
var     exponente     :nodptr;
        matriz3       :matriz_nodos;
 begin
   exponente:=0;    inic1(altura);
   chequeo(matriz1,exponente);
   while not(finalizacion) do
    begin
     exponente:=exponente+1;
     potencia(exponente,matriz3);
     chequeo(matriz3,exponente);
    end;   {while}
  end; {procedure}

{****************** CALCULO DE LOS ACTUALES CANDIDATOS ******************}

{funcion que determina si el elemento pertenece a un arreglo del
tipo conjuntos }

function pertenece (elemento  :nodptr;var arreglo :conjuntos)   :boolean;
var t: boolean;
    k :rango_conj;
begin
 t:=false;  k:=1;
 repeat
   if elemento=arreglo[k] then t:=true;
   k:=k+1;
 until  (arreglo[k]=0) or (t);
 pertenece:=t;
end;  {function}
```

```
{funcion que determina si el elemento pertenece a un arreglo del
tipo conjunto_sol }

function pertenece_sol(elemento:nodptr;var arreglo :conjunto_sol):boolean;
var t: boolean;
    k :rango_conj_sol;
begin
 t:=false;  k:=1;
 repeat
   if elemento=arreglo[k] then t:=true;
   k:=k+1;
 until  (arreglo[k]=0) or (t);
 pertenece_sol:=t;
end;  {function}

{funcion que determina si los antecesores del elemento han sido itinerados
 y por lo tanto es elegible }

function elegible (elemento :nodptr;indice_sol:rango_sol): boolean;
var  t :boolean;
     i :nodptr;
begin
 t:=true;   i:=0;
 repeat
  i:=i+1;
  if matriz1[i,elemento] then
     t:=pertenece_sol(i,solucion[indice_sol].conjunto);
 until (i=num_nodos) or not(t);
 elegible:=t;
end;   {function}

procedure actuales_candidatos
(indice_solucion:rango_sol;var num_actuales:rango_conj);
var    evaluacion       :boolean;
       k                :rango_conj_sol;
       j,l1,l2          :rango_conj;
 begin
  k:=1; num_actuales:=0; l1:=1; l2:=1;
  inic1(actuales);
  inic1(hojas);
  inic1(internos);
  with solucion[indice_solucion] do
   begin
    while conjunto[k] <>0  do
    begin
     if conjunto[k]<>-1 then
      for j:=1 to num_nodos  do
      begin
       evaluacion:=matriz1[conjunto[k],j] and not(pertenece(j,actuales))
       and not(pertenece_sol(j,conjunto)) and elegible(j,indice_solucion);
       if evaluacion then
        begin
         num_actuales:=num_actuales+1;
         actuales[num_actuales]:=j;          {este dato sirve para regla 5'}
         if altura[j]=1  then begin
                              hojas[l1]:=j; l1:=l1+1;
```

```
                                    end
            else  begin
                    internos[12]:=j ;12:=12+1;
                  end;
        end;   {if evaluacion}
      end; {for j}
     k:=k+i;
    end;  {while}
   end; {with}
end;   {procedure}

{***************** CALCULO DE SUCESORES COMUNES **********************}

      {funcion que determina si el elemento ya ha sido analizado
                        como sucesor comun}

function ya_analizado  (elemento :nodptr)    :boolean;
var l :rango_suc;
    t :boolean;
begin
 l:=1; t:=false;
 repeat
   if elemento=sucesor[l].sucesor_comun then t:=true;
   l:=l+1;
 until (sucesor[l].sucesor_comun =0) or t;
 ya_analizado:=t;
end; {function}

     { funcion que examina si el arreglo1 contiene al arreglo2 }
     { *Los arreglos estan ordenados en forma creciente*}

function examina1
(arreglo1,arreglo2:conjuntos; limite1,limite2:rango_conj) :boolean;
var i,j :rango_conj;
    t   :boolean;
begin
 j:=0; i:=0; t:=true;
 repeat
  j:=j+1; i:=i+1;
  if arreglo1[i]<>arreglo2[j] then
   begin
     t:=false;
     if limite1<>limite2 then
      repeat
       i:=i+1;
       if arreglo1[i] = arreglo2[j] then t:=true;
      until (i>=limite1) or t;
   end; {if}
 until (i=limite1) or (j=limite2) or not(t) ;
 if j<>limite2 then t:=false;
 examina1:=t;
end; {function}

       { procedimiento que borra al sucesor i }

procedure eliminar (i :rango_suc);
```

```
var   k  :rango_conj;
begin
 with sucesor[i] do
 begin
  sucesor_comun:=0;    k:=1;
  while (k<=num_de_ant) do
   begin
    antecesor[k] :=0;
    k:=k+1;
   end; {while}
  num_de_ant:=0;
 end; {with}
end; {procedure}

        { procedimiento que copia el sucesor[de] al sucesor[a]    }

procedure copiar (de,a :rango_suc);
var  i   :rango_conj;
begin
 with sucesor[a] do
 begin
  sucesor_comun:=sucesor[de].sucesor_comun;
  num_de_ant :=sucesor[de].num_de_ant;
  for i:=1 to num_de_ant do antecesor[i] :=sucesor[de].antecesor[i];
 end; {with}
end; {procedure}

procedure acomodar_sucesores ( long :rango_suc);
var i,j   :rango_suc;
begin
 i:=0;     j:=1;
 while (i < long) or (sucesor[j].sucesor_comun<>0) do
 begin
  i:=i+1;
  if sucesor[i].sucesor_comun=0 then
   begin
    j:=i+1;
    while ((sucesor[j].sucesor_comun=0) and (j<long)) do j:=j+1;
    copiar(j,i);
    eliminar(j);
   end; {if}
 end; {while}
end; {procedure}

procedure simplicacion_de_sucesores (long :rango_suc);
var   i,j    :rango_suc;
      t      :boolean;
begin
 i:=1;
 repeat
 with sucesor[i] do
 begin
  j:=i+1;  t:=false;
  if sucesor_comun <> 0 then
  repeat
   if num_de_ant>=sucesor[j].num_de_ant then
```

```
   begin
   if examina1(antecesor,sucesor[j].antecesor,
      num_de_ant,sucesor[j].num_de_ant) then eliminar(j);
   end  {if num_de_ant}
   else
   begin
   if examina1(sucesor[j].antecesor,antecesor,
      sucesor[j].num_de_ant,num_de_ant)  then eliminar(i);
   t:=true;
   end;  {else}
   j:=j+1;
  until (j>long) or t;
 end; {with}
 i:=i+1;
 until i=long;
 acomodar_sucesores(long);
end; {procedure}

procedure busqueda_de_sucesores_comunes;
var asig,eliminacion :boolean;      j,ji       :nodptr;
    k,l1             :rango_conj;   i          :rango_suc;
begin
 k:=1; l:=1;
 inicia_sucesor;
 while internos[k] <>0 do
 begin
  eliminacion:=false;
  for j:=1 to num_nodos do
   begin
    l1:=1;      asig:=false;
    if matriz1[internos[k],j] and not(ya_analizado(j)) then
     with sucesor[l] do
      begin
       for j1:=1 to num_nodos do
        begin
         if (matriz1[j1,j]) and (pertenece(j1,internos)) then
           begin
            if l1=1 then sucesor_comun :=j;
            antecesor[l1] :=j1;
            l1:=l1+1;
            num_de_ant:=l1-1;
           end;  {if matriz and pertenece}
         if j1=num_nodos then  asig:=true;
        end;   {for j1}
       if (l1=2) and (eliminacion) then
         begin
          sucesor_comun:=0;
          antecesor[l]:=0;
          asig:=false;
         end; {if l1}
       if l1=2 then eliminacion:=true;
      end;  {with}
     if asig then l:=l+1;
   end;    {for j}
  k:=k+1;
 end;  {while}
```

```
 if l>2 then
 simplicacion_de_sucesores(l-1);
end;  {procedure}

{********************* CALCULO DE ALTERNATIVAS ************************}

   { procedimiento que examinando al sucesor[j] encuentra al antecesor[k]
     que no pertenece a un grupo anterior y se obtiene la respuesta t,
     si t=true entonces se selecciona al grupo}

procedure seleccionar (var j:rango_suc; var k:rango_conj; var t:boolean);
var  i :rango_suc;
begin
 repeat
  i:=0;  t:=true;
  repeat
   i:=i+1;
   if matriz1[sucesor[j].antecesor[k],sucesor[i].sucesor_comun]  then
    t:=false;
  until (i=j-1)  or not(t);
  if not(t) then k:=k+1;
 until (sucesor[j].antecesor[k]=0) or (t);
end; {procedure}

{procedimiento que examina sobre todas las posiciones j de la
alternativa[num] que no exista en las alternativas[i] ya generadas,
alguna que tenga los mismos elementos. Se obtiene la respuesta borrar}

procedure analisis_del_grupo2
(nivel:rango_tiempo;var borrar:boolean; num:rango_alt;
n:arreglo_indices1; alternativa:alternativas);
var  i                                :rango_alt;
     j,l                              :periodo;
     busqueda,eliminar,diferente      :boolean;

begin
 borrar:=false; i:=1;
 repeat {sobre las alternativas i}
  j:=perfil[nivel]; diferente:=false; busqueda:=true;
  repeat {sobre la posicion j de la alternativa i}
   l:=perfil[nivel]; eliminar:=false;
   repeat {sobre los sucesores l}
    if (matriz1[alternativa[num,j],sucesor[n[l]].sucesor_comun]) and
       (matriz1[alternativa[i,j],sucesor[n[l]].sucesor_comun]) then
       eliminar:=true;
    l:=l-1;
   until (l=0) or (eliminar);
   if (l=0) and not(eliminar) then diferente:=true;
   j:=j-1;  busqueda:=busqueda and eliminar;
  until (j=0) or diferente;
  i:=i+1;
 until (i=num) or (busqueda);
 if busqueda then borrar:=true;
end;  {procedure}

procedure generar_alternativas_recursivamente2
```

```
(nivel:rango_tiempo; tamano:periodo; var alternativa:alternativas;
var num_alternativas:rango_alt);
var  i,num                :rango_alt;      n    :arreglo_indices1;
     j                    :periodo;        k    :arreglo_indices2;
     borrar,borrado       :boolean;        t    :arr_bool;

     procedure asignacion_iterativa;
     begin
      borrado:=false;  j:=1;
      if n[1]=1 then alternativa[i,1]:=sucesor[1].antecesor[1]
       else begin
              k[1]:=1;
              seleccionar(n[1],k[1],t[1]);
              if t[1] then alternativa[i,1]:=sucesor[n[1]].antecesor[k[1]]
               else begin i:=i-1; borrado:=true; end;
            end;  {else if n[1]=1}
      if (t[1]) and (perfil[nivel]>1) then
       begin
        repeat  {sobre los n[j]}
         j:=j+1;
         if n[j]<>n[j-1] then k[j]:=1 else k[j]:=k[j-1]+1;
         if (n[j]=1) and (sucesor[n[j]].antecesor[k[j]]<>0) then
           begin
             alternativa[i,j]:=sucesor[1].antecesor[j];
             if j=perfil[nivel] then num:=1;
            end {if}
           else begin
                  if sucesor[n[j]].antecesor[k[j]]<>0 then
                    seleccionar(n[j],k[j],t[j])
                   else t[j]:=false;
                  if t[j] then begin
                               alternativa[i,j]:=sucesor[n[j]].antecesor[k[j]];
                               if j=perfil[nivel] then num:=num+1;
                                  end
                   else if not(borrado) then
                                 begin  i:=i-1; borrado:=true; end;
                end;  {else if n[j]=1}
        until (j=perfil[nivel]) or borrado;
       end; {if not(t[1])
      if (num>1) and not(borrado) then
           analisis_del_grupo2(nivel,borrar,num,n,alternativa);
       if (borrar) and not(borrado) then begin i:=i-1; num:=num-1; end;
      end; {sub procedure asignacion_iterativa}

      procedure combinaciones2(size:periodo);
      var  l                :periodo;
           evaluacion       :boolean;
      begin
       repeat {sobre los sucesores}
        evaluacion:=false;
        if size<> tamano then combinaciones2(size+1)
         else begin
                asignacion_iterativa;
                n[size]:=n[size]+1;
                i:=i+1;
                evaluacion:=false;
```

```
              for l:=1 to size-1 do evaluacion:=evaluacion or not (t[l]);
            end;
      until (sucesor[n[size]].sucesor_comun=0) or (evaluacion);
      n[size-1]:=n[size-1]+1;

      for l:=tamano downto size-1 do n[l]:=n[size-1];
     end; {sub procedure combinaciones2}

begin
 inic3(alternativa); inic5(n); inic6(k);
 i:=1; borrar:=false; num:=0; k[1]:=1;
 for j:=1 to tamano do begin n[j]:=1; t[j]:=true; end;
 combinaciones2(1);
 num_alternativas:=i-1;
end; {procedure}

{************************** REGLA 3 **********************************}

procedure jobs_itinerados
(nivel:rango_tiempo; var indice:rango_sol; var longitud:rango_conj;
var arreglo:conjuntos);
var k,acum :rango_conj_sol;
    m      :rango_conj;
begin
 acum:=perfil_acumulado(nivel);
 with solucion[indice] do
  begin
   k:=0; m:=0;
   repeat
    k:=k+1;
    if (conjunto[k] <> -1) and (conjunto[k] <> 0) then
      begin
        m:=m+1;
        arreglo[m]:=conjunto[k];
      end;
   until k=acum;
   longitud:=m;
  end; {with}
end; {procedure}

procedure cota_conjunto (elemento,numero:rango_conj;
var bound,sup:conjuntos);
var i :rango_conj;
begin
 bound[elemento]:=altura[sup[1]];
 for i:=2 to numero do
  if bound[elemento]<altura[sup[i]] then bound[elemento]:=altura[sup[i]];
end; {procedure}

procedure jobs_sin_itinerar
(nivel:rango_tiempo; indice_solucion:rango_sol; var bound:conjuntos;
var alternativa:alternativas; var num_alternativas:rango_alt);
var arreglo,sup      :conjuntos;       ii     :rango_alt;
    i                :nodptr;          j1     :periodo;
    k,m,j            :rango_conj;      t      :boolean;
begin
 inic1(arreglo);
 jobs_itinerados(nivel-1,indice_solucion,k,arreglo);
```

```
 ii:=0;
 repeat
  ii:=ii+1;
  for j1:=1 to perfil[nivel] do arreglo[k+j1]:=alternativa[ii,j1];
  m:=0; inici(sup);
  for i:=1 to num_nodos do
   begin
    j:=0;
    t:=false;
    repeat
     j:=j+1;
     if i=arreglo[j] then t:=true;
    until (j=k+perfil[nivel]) or t;
    if not(t) then begin
                    m:=m+1;
                    sup[m]:=i;
                   end;
   end;  {for i}
  cota_conjunto(ii,m,bound,sup);
 until ii=num_alternativas;
end;  {procedure}

procedure minima_cota
(var min :nodptr; num_alternativas:rango_alt; var cota:conjuntos);
var i :rango_alt;
begin
 i:=0;
 min:=cota[1];
 repeat
  i:=i+1;
  if min>cota[i] then min:=cota[i];
 until i=num_alternativas;
end; {procedure}

procedure regla_3
(nivel:rango_tiempo; indice_solucion:rango_sol;var alternativa:alternativas;
var num_alternativas:rango_alt; var min:nodptr);
var  i,k          :rango_conj;
     j1           :periodo;
     cota,bound   :conjuntos;
begin
 inici(bound);
 jobs_sin_itinerar(nivel,indice_solucion,bound,alternativa,num_alternativas);
 inici(cota);
 i:=0;
 repeat
  i:=i+1;
  cota[i]:=nivel+bound[i];
 until i=num_alternativas;
 minima_cota(min,num_alternativas,cota);
  {reduccion de alternativas}
 k:=0;
 for i:=1 to num_alternativas do
  if min=cota[i] then begin
                       k:=k+1;
                       for j1:=1 to perfil[nivel] do
```

```
                        alternativa[k,j1]:=alternativa[i,j1];
                    end;
 num_alternativas:=k;
end;  {procedure}

{************************** REGLA 4 **********************************}

     { funcion que examina si el arreglo1 contiene al arreglo2 }

function examina
(arreglo1,arreglo2:conjuntos; limite1,limite2:rango_conj) :boolean;
var i,j :rango_conj;
    t   :boolean;
begin
 j:=0; i:=0; t:=true;
 repeat
  j:=j+1; i:=i+1;
  if arreglo1[i]<>arreglo2[j] then
   begin
     t:=false;
     if limite1 <>limite2 then
      repeat
       i:=i+1;
       if arreglo1[i] = arreglo2[j] then t:=true;
     until (i=limite1) or t;
   end; {if limite}
 until (i=limite1) or (j=limite2) or not(t)   ;
 if j <> limite2 then t:=false;
 examina:=t;
end; {function}

function sol_factible (nivel:rango_tiempo; var indice :rango_sol;
var primer_sol:boolean)  :boolean;  FORWARD;

    { procedimiento que ordena al conjunto de menor a mayor
      con n=elementos a ordenar}

procedure ordena(var a :conjuntos; n :rango_conj);
var pass,j           :rango_conj;
    hold             :nodptr;
    intchnge         :boolean;
begin
 intchnge:=true;
 pass:=1;
 while (pass<=n-1) and (intchnge) do
  begin
   intchnge:=false;
   for j:=1 to n-pass do
    if a[j] > a[j+1] then begin
                            intchnge:=true;
                            hold:=a[j];
                            a[j]:=a[j+1];
                            a[j+1]:=hold;
                          end;
   pass:=pass+1;
  end; {while}
```

```
end; {procedure}

procedure regla_4
(nivel:rango_tiempo; indice_solucion:rango_sol;
var alternativa:alternativas;var num_alternativas:rango_alt);
var longitud,longi          :rango_conj;    indice          :rango_sol;
    k,i                     :rango_alt;     arreglo,arreglo1 :conjuntos;
    j1                      :periodo;       contenido       :arr_bool;
    primer_sol              :boolean;
begin
 inic1(arreglo); inic1(arreglo1);
 indice:=1; primer_sol:=false;
 repeat {sobre todas las soluciones factibles}
  jobs_itinerados(nivel,indice,longitud,arreglo);
  ordena(arreglo,longitud);
  i:=0;
  repeat
   i:=i+1;
   jobs_itinerados(nivel-1,indice_solucion,longi,arreglo1);
   for j1:=1 to perfil[nivel] do arreglo1[longi+j1]:=alternativa[i,j1];
   ordena(arreglo1,longi+j1);
   contenido[i]:=examina(arreglo,arreglo1,longitud,longi+j1);
  until i=num_alternativas;
  { reduccion de alternativas }
  k:=0;
  for i:=1 to num_alternativas do
   if not(contenido[i]) then begin
                               k:=k+1;
                               for j1:=1 to perfil[nivel] do
                                 alternativa[k,j1]:=alternativa[i,j1];
                             end; {if not(contenido)}
  indice:=indice+1;
  num_alternativas:=k;
 until not(sol_factible(nivel,indice,primer_sol));
end;  {procedure}

{********************* REGLA 5' ********************************}

function max_cardinalidad
(var cardinalidad:conjuntos; var num_alternativas:rango_alt)  :rango_conj;
var big :nodptr;
    i   :rango_conj;
begin
 big:=cardinalidad[1];
 for i:=2 to num_alternativas do
  if big< cardinalidad[i] then big:=cardinalidad[i];
 max_cardinalidad:=big;
end; {function}

procedure regla_5_prima
(nivel:rango_tiempo; indice_solucion:rango_sol; var alternativa:alternativas;
var num_alternativas:rango_alt);
var max,i,num_actuales  :rango_conj;  acum          :rango_conj_sol;
    j1                  :periodo;     cardinalidad  :conjuntos;
    j                   :rango_alt;
begin
```

```
 acum:=perfil_acumulado(nivel-1);
 {nivel-1 ya que se busca el ultimo elemento de solucion.conjunto}
 with solucion[indice_solucion] do
  begin
    j:=0;
    repeat
     j:=j+1;
     for j1:=1 to perfil[nivel] do  conjunto[acum+j1]:=alternativa[j,j1];
     actuales_candidatos (indice_solucion,num_actuales);
     cardinalidad[j]:=num_actuales;
    until j=num_alternativas;
    for j1:=1 to perfil[nivel] do  conjunto[acum+j1]:=0;
  end;  {with}
 max:=max_cardinalidad(cardinalidad,num_alternativas);
 { reduccion de alternativas }
 j:=0;
 for i:=1 to num_alternativas do
  if max=cardinalidad[i] then begin
                               j:=j+1;
                               for j1:=1 to perfil[nivel] do
                                alternativa[j,j1]:=alternativa[i,j1];
                              end; {if max}
 num_alternativas:=j;
end; {procedure}

{********************* SCHEDULE ***************************************}

    { funcion que calcula el numero de elementos de un vector tipo
      conjuntos}

function length(x:conjuntos)                    :rango_conj;
var k  :rango_conj;
begin
 k:=1;
 while(x[k]<>0) do k:=k+1;
 length:=k-1;
end; {function}

procedure itin_internos_hojas (nivel:rango_tiempo;
 indice_solucion:rango_sol);
var i,lengint,lenghoj  :nodptr;
    acum,k             :rango_conj_sol;
begin
 lengint:=length(internos);
 lenghoj:=length(hojas);
 acum:=perfil_acumulado(nivel-1)+1;
 with solucion[indice_solucion] do
  begin
   for i:=1 to lengint do
    begin
     conjunto[acum]:=internos[i];
     acum:=acum+1;
    end;  { for i}
   if lenghoj > 0 then
    for i:=1 to perfil[nivel]-length(internos) do
     if hojas[i]<>0 then
       begin
```

```
        conjunto[acum]:=hojas[i];
         acum:=acum+1;
       end; {if hojas}
   k:=0;
   for i:=1 to (perfil[nivel]-lengint-lenghoj) do
   begin
    conjunto[acum]:=-1;
    acum:=acum+1;
    k:=k+1;
   end; {for i}
   num_de_ociosos:=num_de_ociosos+k;
  end;  {with}
end;  {procedure}

procedure itinerar_alternativa
(numero:rango_alt; nivel:rango_tiempo; indice_solucion:rango_sol;
var alternativa:alternativas);
var  i      :periodo;
     acum   :rango_conj_sol;
begin
 acum:=perfil_acumulado(nivel-1);
 with solucion[indice_solucion] do
  for i:=1 to perfil[nivel] do conjunto[acum+i]:=alternativa[numero,i];
end;  {procedure}

procedure secuenciar_hojas
(nivel:rango_tiempo; indice_solucion:rango_sol; var primer_sol :boolean);
var k          :rango_conj;
    acum,i     :rango_conj_sol;
begin
 if (primer_sol) and (indice_solucion=1) then primer_sol:=false;
 k:=0;  acum:=perfil_acumulado(nivel-1);     i:=acum;
 with solucion[indice_solucion] do
   begin
    while (k<>length(hojas))    do
     begin
      k:=k+1;  i:=i+1;
      conjunto[i]:=hojas[k];
     end;
    nivel:=nivel-1;
    while (i>acum) do          {actualizacion del nivel}
     begin
      nivel:=nivel+1;
      acum:=perfil_acumulado(nivel);
     end;
    for k:=i+1 to acum do conjunto[k]:=-1;      {procesadores ociosos}
    cota:=nivel;
    num_de_ociosos:=num_de_ociosos + (acum-i);
   end;  {with}
end;  {procedure}

procedure cambio_de_solucion (nivel:rango_tiempo;
var indice_solucion:rango_sol);
var  i,acum :rango_conj_sol;
begin
 acum:=perfil_acumulado(nivel-1);
```

```
 with solucion[indice_solucion+1] do
  begin
   for i:=1 to acum do
    begin
     conjunto[i]:= solucion[indice_solucion].conjunto[i];
     if conjunto[i]=-1 then num_de_ociosos:=num_de_ociosos+1;
    end; {for i}
  end; {with}
 indice_solucion:=indice_solucion+1;
end; {procedure}

procedure eliminar_solucion (nivel:rango_tiempo;
var indice_solucion:rango_sol);
var i,acum    :rango_conj_sol;
begin
 acum:=perfil_acumulado(nivel);
 with solucion[indice_solucion] do
  begin
   for i:=1 to acum do conjunto[i]:=0;
   num_de_ociosos:=0;
   cota:=0;
  end; {with}
 indice_solucion:=indice_solucion-1;
end; {procedure}

procedure analisis_de_solucion (nivel:rango_tiempo; var indice:rango_sol);
var i  :rango_conj_sol;
begin
 with solucion[i] do
  if (solucion[indice].cota < cota) and
  (solucion[indice].num_de_ociosos < num_de_ociosos) then
   begin
    for i:=1 to perfil_acumulado(nivel) do
     conjunto[i]:=solucion[indice].conjunto[i];
    num_de_ociosos:=solucion[indice].num_de_ociosos;
    cota:=solucion[indice].cota;
    eliminar_solucion(nivel,indice);
   end; {if}
end; {procedure}

function sol_factible
(nivel:rango_tiempo;var indice:rango_sol;var primer_sol:boolean):boolean;
var longitud  :rango_conj;
    arreglo   :conjuntos;
begin
 inici(arreglo);
 jobs_itinerados (max_rango_tiempo,indice,longitud,arreglo);
 if longitud=num_nodos then begin
                             sol_factible :=true;
                             if indice=1 then primer_sol:=false
                             else analisis_de_solucion(nivel,indice);
                            end
 else sol_factible := false;
end;  {function}

procedure Salida (hora,min,seg,dec :word);  forward;
```

```
procedure perfil_insuficiente(nivel:rango_tiempo);
var  w :word; dato,code :integer; s :string;
begin
 clrscr; w:=getcolor; textcolor(red+blink); gotoXY(9,13);
 writeln('EL PERFIL DE TIEMPO ES INSUFICIENTE PARA UNA ITINERACION TOTAL');
 textcolor(w);
 if nivel=max_rango_tiempo then writeln('   PROGRAMA ABORTADO');
 repeat
   gotoXY(9,15);
   write('                                                                    ');
   gotoXY(9,15);
   write('Para continuar dame el valor del Perfil [',nivel,']= ');
   readln(s);  Val(S, dato, code);
   if (dato<1) or (dato>maxtiempo) or (code<>0) then
    begin
     gotoXY(9,17);
     write('          El intervalo válido es [1,',maxtiempo,']');
     delay(3000);  gotoXY(9,17);
     write('                                                    ');
    end; {if dato}
 until (dato>=1) and (dato<=maxtiempo) and (code=0);
 perfil[nivel]:=dato; inc(num_etapas);
end; {procedure}

procedure schedule
(var primer_sol:boolean; nivel:rango_tiempo; var indice_sol:rango_sol);
var    num_actuales  :rango_conj;    i,num_alternativas :rango_alt;
       alternativa   :alternativas;  min                :nodptr;
begin
 if perfil[nivel]=0 then perfil_insuficiente(nivel);
 actuales_candidatos(indice_sol,num_actuales);
 if internos[1]<>0 then
  begin
   if length(internos)>perfil[nivel] then
    begin
     busqueda_de_sucesores_comunes;
     generar_alternativas_recursivamente2
     (nivel,perfil[nivel],alternativa,num_alternativas);
     regla_3(nivel,indice_sol,alternativa,num_alternativas,min);
     if not(primer_sol) and (num_alternativas>1) then
        regla_4(nivel,indice_sol,alternativa,num_alternativas);
     if num_alternativas>1 then
        regla_5_prima(nivel,indice_sol,alternativa,num_alternativas);
     i:=0;
     while (i<>num_alternativas) do
      begin
       i:=i+1;
       if i > 1 then cambio_de_solucion(nivel,indice_sol);
       if indice_sol > maxsoluciones then exit; {Saturacion de soluciones}
       itinerar_alternativa(i,nivel,indice_sol,alternativa);
       if not(primer_sol) and
       ((solucion[1].num_de_ociosos < solucion[indice_sol].num_de_ociosos)
        or (solucion[1].cota < solucion[indice_sol].cota)) then
        begin
         eliminar_solucion(nivel,indice_sol);
```

```
        i:=i+1;
       end {if not}
       else  if not(sol_factible(nivel,indice_sol,primer_sol))
                then  schedule (primer_sol,nivel+1,indice_sol);
     end; {while i}
   end  {if length}
  else begin
        itin_internos_hojas(nivel,indice_sol);
        if not(sol_factible(nivel,indice_sol,primer_sol))
           then schedule(primer_sol,nivel+1,indice_sol);
       end;  {else if length}
 end  {if internos}
else begin
      secuenciar_hojas(nivel,indice_sol,primer_sol);
      analisis_de_solucion(nivel,indice_sol);
     end; {else if internos}
end; {procedure}

{********************* DATOS DE ENTRADA Y SALIDA ************************}

procedure datos_ejemplo;
begin
 num_nodos:=18; inicia_matriz(matriz1);
 matriz1[1,2]:=true; matriz1[1,3]:=true; matriz1[1,4]:=true;
 matriz1[1,5]:=true; matriz1[1,6]:=true;  matriz1[1,7]:=true;
 matriz1[1,8]:=true; matriz1[2,9]:=true; matriz1[3,9]:=true;
 matriz1[4,9]:=true; matriz1[5,10]:=true; matriz1[6,10]:=true;
 matriz1[7,11]:=true; matriz1[8,11]:=true; matriz1[9,12]:=true;
 matriz1[9,14]:=true; matriz1[10,13]:=true; matriz1[11,16]:=true;
 matriz1[12,18]:=true; matriz1[13,14]:=true;  matriz1[13,15]:=true;
 matriz1[16,17]:=true; matriz1[17,18]:=true;
 num_etapas:=14; inic2(perfil);
 perfil[1]:=2;  perfil[2]:=3; perfil[3]:=4; perfil[4]:=5; perfil[5]:=7;
 perfil[6]:=6; perfil[7]:=2; perfil[8]:=3; perfil[9]:=3; perfil[10]:=4;
 perfil[11]:=2; perfil[12]:=2; perfil[13]:=2; perfil[14]:=2;
 calculo_de_altura;
end;  {procedure}

procedure lectura_de_datos ( var matriz :matriz_nodos);
var i,k,j,num_preced  :nodptr;     correcto    :char;
    s                 :string;     dato,code   :integer;
begin
 clrscr; inicia_matriz(matriz);  inic2(perfil);
 repeat
   repeat
    write('     Cantidad de nodos de la gráfica de precedencia: ');
    readln(s);   Val(S, dato, code);
    if (dato<0) or (dato>maxnodos) or (code<>0) then
     writeln('       El intervalo válido es [0,',maxnodos,']');
   until (dato>0) and (dato<=maxnodos) and (code=0);
   num_nodos:=dato;
  for i:=1 to num_nodos do
   begin
    repeat
     write('     Número de nodos que precede el nodo( ',i,' )');
     readln(s); Val(S, dato, code);
```

```
    if (dato<0) or (dato>num_nodos) or (code<>0) then
     writeln ('          El intervalo válido es [0,',num_nodos,']');
   until (dato>=0) and (dato<=num_nodos) and (code=0);
   num_preced:=dato;
   for j:=1 to num_preced do
    begin
     repeat
      write('       El nodo ( ',i,' ) precede al nodo : ');
      readln(s); Val(S, dato, code);
      if (dato<=0) or (dato>num_nodos) or (code<>0) then
       writeln('          El intervalo válido es [1,',num_nodos,']');
     until (dato>0) and (dato<=num_nodos) and (code=0);
     k:=dato;
     matriz[i,k]:=true;
    end;
  end;  {for i}
 writeln; writeln;
 write('¿ Están los datos correctos ?, (S/N)  '); readln(correcto);
 if (correcto='S') or (correcto='s') then
  begin
   repeat
    clrscr;writeln;write('     Número de etapas del perfil de tiempo:  ');
    readln(s); Val(S, dato, code);
    if (dato<1) or (dato>max_rango_tiempo) or (code<>0) then
     writeln('          El intervalo válido es [1,',max_rango_tiempo,']');
   until (dato>=1) and (dato<=max_rango_tiempo) and (code=0);
   num_etapas:=dato;
   for i:=1 to num_etapas do
    begin
     repeat
      write('       perfil( ',i,') : ');
      readln(s);  Val(S, dato, code);
      if (dato<0) or (dato>maxtiempo) or (code<>0) then
       writeln('          El intervalo válido es [0,',maxtiempo,']');
     until (dato>=0) and (dato<=maxtiempo) and (code=0);
     perfil[i]:=dato;
    end; {for i}
   writeln; writeln;
   write('¿ Están los datos correctos ?, (S/N)  '); readln(correcto);
  end; {if correcto}
 clrscr;
until (correcto='S') or (correcto='s');
calculo_de_altura;
end; {procedure}

procedure salida;
var  i   :rango_sol;          m   :rango_conj_sol;
     j   :rango_tiempo;       k   :periodo;
     t   :boolean;
begin
 i:=1;  t:=true;
 repeat
  clrscr;
  with solucion[i] do
  begin
   m:=i;
```

```
    for j:=1 to num_etapas do
    begin
     write('   En el Perfil [',j,']   (');
     for k:=1 to perfil[j] do
     begin
      if (conjunto[m]=-1) or (conjunto[m]=0) then write('*,')
       else write(conjunto[m],',');
      m:=m+1;
     end; {for k}
     writeln(')');
    end; {for j}
    writeln;
    writeln('     El número de procesadores ociosos es: ', num_de_ociosos);
    writeln;
    writeln('     El número de etapas empleadas es: ',cota);
    writeln;
    writeln('Solución obtenida en un tiempo de computadora de: ');
    writeln('    ',hora,' hr. ',min,' min. ',seg,' seg. ',dec,' dec.' );

   end; {with}
   i:=i+1;      readln;
  until not(sol_factible(max_rango_tiempo,i,t));
 end; {procedure}

{************************** GRAFICACION ********************************}

 procedure modo_grafico;
 var  grDriver,grMode,ErrCode       :Integer;
 begin
  grDriver := Detect;
  InitGraph(grDriver,grMode,'a:');
  ErrCode := GraphResult;
  if ErrCode <> grOk then
   begin
    clrscr; gotoXY(30,13); sound(400);delay(200);Nosound;
    WriteLn('Error de Graficación:',GraphErrorMsg(ErrCode));
   end;{if}
 end; {procedure}

procedure grafica_de_precedencia (recorrido_opt:boolean);

var  num_de_elementos,generacion,niveles                    :rango_conj;
     indice,j                                               :rango_conj_sol;
     nivel                                                  :rango_tiempo;
     renglon,sum_renglon,columna,maxcolumna,aux,radio       :integer;
     x,y                                                    :renglones;
     save                                                   :conjunto_sol;

        { funcion que calcula el numero de alturas diferentes }

 function niveles_de_altura( altura:conjuntos)  :rango_conj;
 var  i,n  :rango_conj;
 begin
  ordena(altura,num_nodos);
  n:=1;
  for i:=1 to num_nodos-1 do
```

```
  if altura[i] <> altura[i+1] then n:=n+1;
 niveles_de_altura:=n;
end; { sub function}

       { funcion que convierte del tipo entero al tipo string }

function IntToStr(i: Longint): string;
 var
   s: string[2];
 begin
   Str(i, s);
   IntToStr := s;
 end; { sub function}

procedure salvar_sol;
var  i :rango_conj_sol;
begin
 for i:=1 to max_rango_conj_sol do save[i]:=solucion[1].conjunto[i];
end; {sub procedure}

procedure recorrer(radio:integer);
var  i,k :rango_conj_sol;   num :nodptr;   nivel :rango_tiempo;
begin
 i:=0; num:=0; nivel:=1;
 repeat
  i:=i+1;
  if (save[i]<>-1) and (save[i]<>0) then
   begin
     num:=num+1; k:=1;
     while(save[i]<>solucion[1].conjunto[k]) do k:=k+1;
     circle(x[k],y[k],radio+10);
   end;
  if i=perfil_acumulado(nivel) then begin inc(nivel);delay(3000); end;
 until num=num_nodos;
end; {procedure}

procedure graficar_primera_generacion
(var nivel:rango_tiempo; renglon,columna,radio:integer);
var  num_de_elementos    :rango_conj_sol;
     i                   :rango_suc;
begin
 primera_itineracion (num_de_elementos,nivel);
 x[1]:= columna div (num_de_elementos+1);
 y[1]:=renglon;
 for i:=1 to num_de_elementos do
  begin
   indice:=indice+1;
   circle(x[indice],renglon,radio);
   outtextXY(x[i],renglon-2,inttostr(solucion[1].conjunto[indice]));
   x[indice+1]:=x[indice] + x[1];   y[indice+1]:=renglon;
  end; {for i}
end; { sub procedure}

procedure lineas;
var   j    :nodptr;
      i,k  :rango_renglon;
```

```
begin
  i:=1; nivel:=1;
  while(solucion[1].conjunto[i] <> 0) do
  begin
   for j:=1 to num_nodos do
    if matriz1[solucion[1].conjunto[i],j] then
     begin
      k:=1; while (j<>solucion[1].conjunto[k]) do k:=k+1;
      line(x[i],y[i]+(radio-3),x[k],y[k]-(radio-3));
     end; {if matriz1}
    i:=i+1;
  end;  {while}
end; { sub procedure}

procedure itinerar_actuales_candidatos (var n :rango_conj_sol);
var k    :rango_conj;
begin
 with solucion[1] do
  begin
   k:=1;
   while (internos[k] <> 0) do
    begin
     n:=n+1;
     conjunto[n]:=internos[k];
     k:=k+1;
    end;  {while internos}
   k:=1;
   while (hojas[k] <> 0) do
    begin
     n:=n+1;
     conjunto[n]:= hojas[k];
     k:=k+1;
    end; {while hojas}
  end; {with}
end;  { sub procedure}

procedure impresion_generacion (renglon,anterior,columna,radio:integer);
var   i      :rango_suc;
      aux    :integer;
begin
 i:=0;
 x[indice]:= columna;
 aux:=x[indice];
 repeat
  if solucion[1].conjunto[indice]<>-1 then
   begin
    y[indice]:=renglon;
    circle(x[indice],y[indice],radio);
    outtextXY(x[indice],renglon-2,inttostr(solucion[1].conjunto[indice]));
    x[indice+1]:=x[indice]+aux;
    i:=i+1;
   end {if solucion}
  else x[indice+1]:=x[indice];
  indice:=indice+1;
 until (i=num_de_elementos);
end; { sub procedure}
```

```
begin
 inic4(x); inic4(y); radio:=20;  indice:=0; modo_grafico;
 niveles:=niveles_de_altura(altura);
 renglon:=getmaxY div (niveles+1);
 sum_renglon:=renglon;  maxcolumna:= getmaxX;
 if recorrido_opt then salvar_sol;
 graficar_primera_generacion(nivel,renglon,maxcolumna,radio);
 generacion:=1;  j:=perfil_acumulado(nivel); indice:=indice+1;
 repeat
  generacion:=generacion+1;  aux:=sum_renglon;
  sum_renglon:=renglon + sum_renglon;
  actuales_candidatos(1,num_de_elementos);
  itinerar_actuales_candidatos (j);
  columna:= maxcolumna div (num_de_elementos+1);
  impresion_generacion(sum_renglon,aux,columna,radio);
 until generacion=niveles;
 lineas;
 if recorrido_opt then recorrer(radio);
 while not(keypressed) do; readln;  closegraph;
end;{procedure}

{********** CONTROL Y EJECUCION DE SCHEDULE ********************}

procedure control_de_ejecucion (ejemplo:boolean);
var   k                                :rango_conj_sol;
      nivel                            :rango_tiempo;
      t,primer_sol                     :boolean;
      optima,indice_solucion           :rango_sol;
      h1,h2,m1,m2,s1,s2,hund1,hund2 : Word;
begin
   clrscr;
    indice_solucion:=1; primer_sol:=true;
    if not(ejemplo) then begin
                           lectura_de_datos(matriz1);
                           grafica_de_precedencia(false);
                         end;
    getTime(h1,m1,s1,hund1);
    primera_itineracion(k,nivel);    { k no sirve aqui}
    nivel:=nivel+1;
    if not(sol_factible(nivel,indice_solucion,primer_sol)) then
      schedule(primer_sol,nivel,indice_solucion);
    getTime(h2,m2,s2,hund2);
    salida (h2-h1,m2-m1,s2-s1,hund2-hund1);
    clrscr; gotoXY(10,13); write('EL RECORRIDO OPTIMO ES EL SIGUIENTE:');
    readln;    grafica_de_precedencia(true);
end; {procedure}

{************************** PRESENTACION ***************************}

procedure Informacion;
var s  :string;  f :text;  i :rango_conj_sol;
begin
 assign(f,'a:inform.txt'); reset(f); clrscr; i:=0;
 while not(EOF(f)) do
  begin
```

```
    readln(f,s); writeln(s); delay(1000); inc(i);
    if i=24 then begin
                   writeln('    "ENTER"  PARA CONTINUAR');
                   readln;
                 end;
  end; {while}
 writeln(' FIN DE INFORMACION. "ENTER PARA CONTINUAR"'); readln;
 close(f);
end; {procedure}

procedure Ejemplo;
var i    :nodptr;
begin
 clrscr;
 gotoxy(10,13); write('SI SE TIENE LA SIGUIENTE GRAFICA DE PRECEDENCIA:');
 datos_ejemplo;
 grafica_de_precedencia(false);
 gotoxy(7,3); write('CON EL SIGUIENTE PERFIL DE TIEMPO:');
 for i:=1 to num_etapas do begin
                    gotoxy(35,5+i);writeln('Perfil [',i,']=  ',perfil[i]);
                   end;
 readln;  clrscr;
 gotoxy(10,13); write('AL HACER LA ITINERACION SE OBTIENE:'); readln;
 control_de_ejecucion(true);
end; {procedure}

procedure pantallas;
var s          :string[10];
    i, code    :Integer;
begin
 clrscr; textbackground(blue); textcolor(magenta);
 gotoxy(21,6);write('UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO');
 gotoxy(20,12);write('ESCUELA NACIONAL DE ESTUDIOS PROFESIONALES');
 gotoxy(18,18);write('LICENCIATURA EN MATEMATICAS APLICADAS Y COMPUTACION');
 delay(4000); clrscr;
 gotoxy(27,6);write('TESIS PROFESIONAL TITULADA');
 gotoxy(20,11);write('"APLICACION DE LA TECNICA DE RAMIFICACION');
 gotoxy(20,16);write(' Y ACOTACION A UN PROBLEMA DE ITINERACION" ');
 gotoxy(23,21);write('DEL ALUMNO: EDUARDO SCHMIDKONZ GARCIA');
 delay(4000);
 repeat {regreso al menu principal}
  clrscr;
  gotoxy(33,4);write('MENU PRINCIPAL');
  gotoxy(20,8);write('1- INFORMACION GENERAL DEL PROGRAMA');
  gotoxy(20,12);write('2- EJEMPLO ILUSTRATIVO ');
  gotoxy(20,16);write('3- EJECUCION DEL PROGRAMA');
  gotoxy(20,20);write('4- SALIDA DEL PROGRAMA');
  gotoxy(60,23);write('OPCION: ');
  repeat
   readln(s);
   Val(S, i, code);
   if (i<1) or (i>4) or (code<>0) then
    begin
      gotoxy(50,24); write('OPCION INCORRECTA');
      delay(1000);
      gotoxy(50,24); write('                   ');
```

```
      gotoxY(68,23);write('                    ');
      gotoxY(69,23);
    end;{if}
  until (i>=1) and (i<=4);
  case i of
    1: Informacion;
    2: Ejemplo;
    3: Control_de_ejecucion(false);
    4: begin clrscr; gotoxy(35,13); write('HASTA PRONTO');halt; end;
  end; {case of}
 until i=4;
end;  {procedure}

 {************************* PROGRAMA PRINCIPAL ***************************}

begin
 pantallas;
end.  {program}
```

APENDICE B.

LA TERMINOLOGIA DE LA TEORIA DE ITINERACION.

En este apéndice, se presenta la terminología más común que se aplica en la teoría de itineración, con el objeto de tener un conocimiento más preciso de los términos que se emplean, así como de los sinónimos que existen. Con la finalidad de tener un fundamento para futuras investigaciones.

Inventario de espera. Son los trabajos en procesamiento, esperando a ser realizados en las máquinas. Un trabajo puede estar en el inventario de espera en cualquier momento entre el comienzo y final de su procesamiento.

Itinerario. Es una secuencia factible, en la cual los tiempos de inicio y terminación de operaciones que comprenden todas los trabajos en cada máquina, están especificados. También se especifica el tiempo de ocio, si existe, entre el periodo de procesamiento de dos operaciones sucesivas.

Máquina. Es un mecanismo simple capaz de ejecutar un cierto proceso. Un nombre alterno es: procesador, recurso o centro de trabajo.

Operación. Es un trabajo elemental que define a un trabajo más complejo. Una operación está especificada enteramente por el trabajo y la máquina involucrados. Un nombre alterno es: actividad.

Orden de máquina. Es un arreglo de un conjunto de máquinas a través del cual un trabajo en particular va a ser ejecutado, dependiendo primeramente de los requerimientos tecnológicos. Este orden, generalmente se encuentra descrito por las relaciones de precedencia de sus operaciones. Un nombre alterno es: rutina, ordenamiento u orden tecnológico.

Tasa de producción. Es la proporción en que el trabajo fluye a través del sistema operacional. El trabajo puede ser medido en varias unidades como números, peso, volúmen, tiempo o valor. Un nombre alterno es: tasa de salida.

Tiempo de espera de trabajo. Es la cantidad de tiempo que un trabajo tiene que esperar antes de empezar su siguiente operación, en una máquina en particular, de acuerdo a su orden tecnológico.

Tiempo de flujo de trabajo. Es la cantidad de tiempo que un trabajo

gasta en el sistema. Un nombre alterno es: tiempo de residencia, tiempo de sistema o intervalo de manufactura.

Tiempo de itinerario. Es el intervalo de tiempo entre un tiempo de liberación de un conjunto de trabajos en un sistema y su finalización. Puede ser interpretado como la fecha de terminación del último trabajo de un conjunto. Un nombre alterno es: tiempo máximo de terminación, makespan, o tiempo total transcurrido.

Tiempo de liberación de trabajo. Es el tiempo en el cual un trabajo es liberado al sistema después de que ha sido planeado. Un nombre alterno es: tiempo de disponibilidad de trabajo ó tiempo de arribo de trabajo.

Tiempo de ocio de máquina. Es la cantidad de tiempo que una máquina no es utilizada antes de ejecutar una operación.

Tiempo de proceso. Es el tiempo requerido para completar una operación en una máquina en particular. Puede incluir el tiempo de disposición de máquina, el tiempo de transportar el trabajo asociado de una máquina a otra, y/o el tiempo de readaptación de la máquina después de otra operación. Un nombre alterno es: tiempo de operación o tiempo de corrida.

Tiempo de terminación de trabajo. Es el tiempo en que la última operación de un trabajo específico es terminada.

Tiempo de vencimiento. Es el tiempo en el cual un trabajo debería de ser completado y entregado. Es equivalente al tiempo en el cual la última operación de un trabajo debería ser completada. También puede ser definido como el tiempo deseable de terminación.

Tiempo permitido de trabajo. Es el tiempo permitido a un trabajo para estar en el sistema. Se puede interpretar como la diferencia entre el tiempo de vencimiento y el tiempo de liberación de un trabajo.

Trabajo. Es una unidad de un producto o un conjunto idéntico de unidades, que deben ser procesadas en ciertas máquinas. Un nombre alterno es: tarea, lote, lote de trabajo, lote de producción, corrida de producción u orden de sistema.

Trabajo adelantado. Es la cantidad de tiempo en el que un trabajo es terminado con respecto a su vencimiento. Este considera sólo la diferencia negativa entre el tiempo de terminación y el vencimiento de un trabajo.

Trabajo retardado. Es la diferencia algebraica entre el tiempo de

terminación y el vencimiento de un trabajo.

Trabajo retrasado. Es la cantidad de tiempo en que un trabajo es terminado después de su vencimiento. El retraso considera sólo diferencias positivas entre el tiempo de terminación y el vencimiento de un trabajo.

Secuencia. Es un conjunto de secuencias de trabajos, cada una asignada a cada máquina. La secuencia generalmente especifica el arreglo de operaciones que comprenden todos los trabajos en todas las máquinas. Una secuencia, no señala el tiempo en el cual las operaciones son ejecutadas, ni la existencia de tiempos de ocio entre varias operaciones.

Secuencia de trabajo. Es un arreglo de un conjunto de trabajos a ser procesados en una máquina simple. Esta secuencia está usualmente descrita por las relaciones de precedencia entre el conjunto de trabajos en esa máquina.

Utilización de sistema. Es la relación total de trabajo cargado y la capacidad disponible de las máquinas del sistema en un período dado de itinerario. El trabajo total cargado puede ser expresado en términos del tiempo total de procesamiento a ser itinerado en todas las máquinas. Similarmente, la capacidad disponible de máquina puede ser expresada en términos del tiempo a ser empleado para ejecutar el trabajo. La utilización de sistema es por lo tanto, una fracción de la capacidad disponible de máquina que es empleada para ejecutar un trabajo cargado asignado. Un nombre alterno es: utilización de máquina.

BIBLIOGRAFIA.

1. Agin, N., (1966) "Optimum seeking with branch and bound", Management Science, Vol.13, No.4, B:176 a B:185.

2. Ashour, S., (1972) "Sequencing theory", Springer Verlag, Germany.

3. Baker, K., (1974) "Introduction to sequencing and scheduling", John Wiley & Sons, USA.

4. Balas, E., (1968) "A note on the Branch and Bound Principle", Operations Research. Vol 16, No. 2, 442 a 444. Errata, (1968) Vol.16, No.4, 886.

5. Brooks, G.R. y White, C.R., (1965) "An algorithm for finding optimal or near optimal solution to the production scheduling problem", Journal of industrial engineering, Vol.16, No.1.

6. Brucker, P., Garey, M.R., y Johnson, D.S., (1977) "Scheduling equal-length task under tree-like precedence constraints to minimize maximum lateness", Mathematics and Operations Research, Vol.2, 275 a 284.

7. Coffman, E.G., et al, (1976) "Computer and job-shop scheduling theory", John Wiley and Sons, USA.

8. Conway, R.W., Maxwell, W.L., y Miller, L.W., (1967) "Theory of scheduling", Adison-Wesley, Reading, Mass.,USA.

9. Dolev, D. y Warmuth, M.K. (1984) "Scheduling precedence graphs of bounded height", Journal of Algorithms, Vol.5, 48 a 59.

10. Dolev, D., Upfal, E. y Warmuth, M.K., (1985) "Scheduling trees in parallel", VLSI: Algorithms Architecture. 91 a 102.

11. Fillmore, J.P. y Williamson, S.G. (1974) "On backtracking: A combinatorial description of the algorithm", SIAM Journal of Computers, Vol.3, 41 a 55.

12. Garey, M.R., (1973) "Optimal task sequencing with precedence constraints", Discrete mathematics,Vol.4, 37 a 56.

13. Garey, M.R. y Graham, R.L, (1973) "Bounds on scheduling

with limited resources", Operation Systems Rev. Vol.7, 104 a 111.

14. Garey, M.R. y Graham, R.L, (1975) "Bounds for multiprocessing scheduling with resource constraints", SIAM Journal of Computers, Vol.4, 187-200

15. Graham, R.L., Lawler, E.L., Lenstra, J.K. y Rinnooy Kan, A.H.G., (1979) "Optimization and aproximation in deterministic sequencing and scheduling: A survey", Ann. Discrete Mathematics, Vol.5, 287-326.

16. Hu, T.C., (1961) "Parallel sequencing and assembly line problems", Operations Research,Vol.9, No.6, 841-848.

17. Hu, T.C., (1982) "Combinatorial algorithms", Addison-Wesley, Reading, Mass. USA.

18. Ibaraki, T., (1976) "Theorical comparison of search strategies in branch and bound algorithms", International Journal of Computing Information Science, Vol.5, No.4, 315 a 344.

19. Ignall, E. y Schrage, L., (1965) "Aplication of the branch and bound technique to some flow.shop problems", Operations Research, Vol.13, No.3, 400-412.

20. Kohler, W.H. y Steiglitz, K., (1974) "Characterization and theorical comparison of branch and bound algorithms for permutation problems", Journal of ACM, Vol.21, No.1, 140 a 156.

21. Lawler, E.L y D.E. Wood, (1966) "Branch and bound methods: a survey", Operations Research, Vol.14, 699-719.

22. Lenstra, J.K., Rinnooy Kan, A.H.G. y Brucker, P., (1977) "Complexity of machine scheduling problems", Ann. Discrete Mathematics, Vol.1, 343-362

23. Lenkstra, J.K. y Rinnooy Kan, A.H.G., (1978) "Complexity of scheduling under precedence constraints", Operations Research, 26, 22-35.

24. Lloyd, E.L. (1982) "Critical path scheduling with resource and processors constraints", Journal of the ACM 29, 781 a 811.

25. Lomnicki, Z., (1965) "A branch and bound algorithm for the

exact solution of the three-machine scheduling problem", Operational Research Quaterly, vol 16, no 1.

26. Mitten, L.G., (1970) "Branch and bound methods: General formulation and properties", Operation Research, Vol.18, No.1, 24 - 34.

27. Reingold, E.M., Nievergelt, J. y Deo, N. (1977) "Combinatorial algorithms: Theory and Practice", Prentice-Hall, N.J.,USA.

28. Simons, B., (1983) "Multiprocesor scheduling of unit-time jobs with arbitrary release times and deadlines", SIAM Journal of Computers, Vol.12, 294-299.

29. Smith, D.R., (1984) "Random trees and the analysis of branch and bound procedures", Journal of the ACM, Vol.31, 163 a 188.

30. Thompson, G.L., Wuth, J.F., et al, (1963) "Industrial scheduling", Prentice Hall, USA.

31. Ullman, J.D., (1975) "NP-Complete scheduling problems", Journal of Computing System Science, Vol.10, 384-393.

32. Yang, Cheng-I.,Wang, Jia-Shung. y Lee, R.C.T., (1989) "A branch and bound algorithm to solve a equal-execution-time job scheduling problem with precedence constraint and profile", Computers and Operation Research, Vol.16, No.3, 257-269.