Centro de Investigación y de Estudios Avanzados
del Instituto Politécnico Nacional

**Unidad Zacatenco**

**Departamento de Computación**

# Enfoque de Programación Lineal Entera Binaria al Problema de la Asignación de la Planificación Multiprocesador

Tesis que presenta

**Liliana Puente Maury**

Para Obtener el Grado de

**Maestro en Ciencias**

En la Especialidad de

**Computación**

Directores de la Tesis:

**Dr. Pedro Mejía Alvarez**

**Dr. Luis Eduardo Leyva del Foyo**

**México, D.F.** **Enero 2012**

*A la memoria de mis abuelos,*
*Bartolomé Maury Duchenes,*
*y Esther Ofelia Guernica Portuondo.*

# Resumen

Recientemente las arquitecturas multiprocesador se han ido popularizando, pues el uso de un sólo procesador no es suficiente para cubrir las demandas computacionales de las aplicaciones modernas. Además, con los límites actuales del hardware, es más fácil y barato incrementar la capacidad de cálculo con un multiprocesador que realizarlo con sistemas uniprocesadores de equivalente capacidad. En los sistemas multiprocesador de tiempo real, es crucial verificar el cumplimiento de los plazos de las tareas, y de esto se ocupa la planificación. La planificación uniprocesador ha sido ampliamente estudiada, pero existen muchos problemas abiertos en la planificación multiprocesador, la cual ha sido abordada principalmente con algoritmos aproximados eficientes.

En este trabajo de tesis, se trató el problema de planificar fuera de línea un conjunto de tareas periódicas sin restricciones de dependencia y que no comparten recursos en $m$ procesadores homogéneos usando el algoritmo de planificación EDF (*Earliest Deadline First*) y bajo un esquema particionado. El problema de asignar las tareas a los procesadores se modeló como un problema de programación lineal entera binaria, y luego se resolvió aplicando la versión del algoritmo de Balas dada por Geoffrion, optimizada para el problema de la planificación de tiempo real usando conocimiento previo del problema. Se evaluó, con resultados experimentales, la factibilidad de la aplicación de este algoritmo exacto de optimización entera para hallar la solución óptima al problema de la planificación multiprocesador, para varios sistemas de tareas en sistemas multiprocesador de tamaño típico. Para algunos de los experimentos, se mostró que es perfectamente posible utilizar este algoritmo fuera de línea para hallar la planificación óptima. Además, a partir de los experimentos realizados es posible hacer conjeturas acerca de las cotas de utilización para la planificabilidad de un conjunto de tareas que cumpla con los requisitos del modelo de tareas propuesto, bajo EDF usando esquema particionado en un sistema multiprocesador.

# Abstract

Recently, multiprocessor systems have become popular, since the use of only one processor is not sufficient to satisfy the computational demands of modern applications. Besides, with the actual hardware limitations, it is easier and cheaper to increase the computational capacity with a multiprocessor than to make it with uniprocessors systems of equivalent capacity. In real time multiprocessor systems, it is crucial to verify the fulfillment of the task deadlines, and this is what scheduling is all about. Uniprocessor scheduling has been widely studied, but there are still open problems in multiprocessor scheduling, which has been faced mainly with efficient approximate algorithms.

In this work, we dealt with the off-line scheduling of a set of periodic tasks with no dependence relations that do not share any resources, using EDF (*Earliest Deadline First*) scheduling algorithm under a partitioned scheme. The allocation problem was modeled as a binary integer linear programming problem, and it was solved by applying Geoffrion's version of Balas algorithm, optimized for the real-time scheduling problem, by using previous knowledge of the problem. The feasibility of applying this integer optimization exact algorithm to find the optimal solution to the multiprocessor scheduling problem, was evaluated, with experimental results, for several task systems in multiprocessor systems of typical size. For some experiments, it was shown that it is perfectly possible to use this algorithm off-line to find the optimal scheduling. In addition, from the experiments that were made, it is possible to make conjectures about the worst case utilization bounds for the schedulability in a multiprocessor system under EDF using the partitioned scheme, for any task set that fulfills the requirements of the proposed task model.

# Agradecimientos:

Deseo expresar mi gratitud:

A mis asesores los Dres. Pedro Mejía Alvarez y Luis Eduardo Leyva del Foyo por su valiosa guía a lo largo de estos dos años.

A mi familia, por su afecto y apoyo en cada momento; en especial a mi esposo Luis, que siempre me ha apoyado y animado en los momentos más difíciles, y a mis padres, que a pesar de la distancia, siempre han velado por mí.

A todos los doctores que han contribuido a mi formación en este centro de estudios, principalmente al Dr. Oliver Schutze y al Dr. Debrup Chakraborty, por haber aceptado revisar este trabajo de tesis.

A mis amigos William Martínez Cortés, Asdrúbal López Chau, Abel Maury Díaz y al Dr. Arnoldo Díaz Ramírez, por sus diversas y muy valiosas contribuciones a este proyecto de tesis.

A mis amigos Eduardo, Joel, Ivonne, Rafa, Lucy y Kimberly, por ofrecerme su amistad y animarme siempre a no flaquear.

A todos mis compañeros de curso, por haber hecho de mi estancia en México un recuerdo inolvidable.

A las secretarias Sofía Reza y Felipa Rosas, por su apoyo invaluable a nosotros los estudiantes.

Al Consejo Nacional de Ciencia y Tecnología (CONACyT) por el apoyo económico otorgado con su programa de becas, y al Centro de Investigación y Estudios Avanzados del IPN (CINVESTAV-IPN) por acogerme como estudiante de su programa de maestría en Computación.

A todos los que de una forma u otra, han contribuido a la elaboración de este proyecto, gracias, muchas gracias.

# Índice general

# Índice de figuras

# Índice de tablas

# Capítulo 1

# Introducción

## 1.1. Sistemas de Tiempo Real

En la actualidad, está presente cada vez más la frase *sistema embebido*. Estos son sistemas autónomos, basados en un microprocesador, que incluyen tanto hardware como software, y cuya función es controlar una o más funciones. Tienen dos características principales:

1. Están dedicados a tareas específicas.
2. Están sujetos a restricciones específicas como: tamaño, peso, consumo energético, etc.

Los sistemas embebidos controlan muchos dispositivos de uso común, implicados en toda la vida moderna. Pueden ser tan pequeños como un reloj digital o una memoria USB y tan grandes como satélites o sistemas controladores de plantas nucleares. Su complejidad también varía desde un único chip hasta redes y periféricos conectados entre sí.

Los sistemas modernos de telecomunicaciones mediante satélites y redes de fibra óptica que trasmiten grandes volúmenes de información en múltiples medios, el control de los procesos industriales y las fábricas en general, los aeropuertos, automóviles, el control del tráfico aéreo y vehicular, los modernos equipos electrónicos (DVDs, consolas de videojuegos, PDAs, impresoras, cámaras digitales, hornos microondas, lavadoras, sistemas de seguridad caseros, reguladores de temperatura), equipos médicos, sistemas de monitoreo de pacientes, son ejemplos de estos sistemas embebidos que han cobrado una importancia capital tanto en la infraestructura tecnológica de la sociedad, como en el aumento de la calidad de vida y la seguridad del hombre.

Todos estos sistemas requieren de varios aspectos como la confiabilidad y la tolerancia a fallos. Algunos de ellos deben funcionar ininterrumpidamente por años sin experimentar fallos o recuperarse de éstos por sí mismos si algún fallo ocurriera. Los sistemas embebidos de seguridad crítica, como los de navegación aérea, los sistemas

de control de reactores, o de fábricas de químicos, requieren de estas características por razones de seguridad. Otros sistemas causarían grandes pérdidas monetarias si fallaran, como por ejemplo, los sistemas de transferencia de fondos bancarios y de mercadeo.

La mayoría de los sistemas embebidos están sujetos a restricciones de tiempo. Es decir, que la correctitud de una operación no depende solamente de su correctitud lógica, sino también del lapso de tiempo en que el sistema sea capaz de completarla. Estos sistemas embebidos se denominan *sistemas embebidos de tiempo real* y se dividen en dos categorías principales, que se citarán a continuación.

En un sistema de tiempo real **crítico** (*hard real time system*), si una tarea completa su ejecución después de su plazo, es lo mismo que si no la completara nunca. Una pérdida de plazos en un sistema de este tipo, puede causar catástrofes o incluso, pérdida de vidas humanas. Ejemplos de sistemas de tiempo real críticos son el sistema de frenado de un auto y el sistema de control de tráfico aéreo. En estos sistemas, es de vital importancia que absolutamente todas las tareas cumplan con sus restricciones de tiempo.

En un sistema de tiempo real **acrítico** (*soft real time systems*), aunque es preferible que las tareas completen su ejecución en menor tiempo del que especifican sus plazos, las restricciones de tiempo no son tan rígidas como en los sistemas de tiempo real críticos mencionados en el párrafo anterior. Por ejemplo, un sistema de compresión de video puede congelar la imagen durante unos cuadros si no se puede ejecutar el algoritmo de descompresión a tiempo, y esto no traerá grandes consecuencias. En otras palabras, que el sistema puede continuar operando aún cuando algunos plazos se hayan incumplido. En este tipo de sistemas, se busca cumplir tantos plazos como sea posible.

En cuanto al cumplimiento de los plazos, en algunos sistemas se dice que la ejecución de las tareas tiene un plazo **firme** si la respuesta del sistema es inútil o contraproducente si se rebasa el plazo de respuesta. En cambio, los sistemas de plazo **flexible** admiten un retraso en el cumplimiento de los plazos, aunque el valor de la respuesta del sistema disminuye progresivamente a medida que pasa el tiempo. Por ejemplo, en una predicción metereológica de un huracán, se define un plazo para obtener la predicción; pero a medida que pase el tiempo y ésta no se obtenga, tendrá mucho menor valor al aumentar el retraso en la información.

Recientemente las arquitecturas multiprocesador se han ido popularizando. Su uso va desde las computadoras personales hasta grandes servidores. Consideraremos aquí un sistema multiprocesador como un ordenador con un conjunto de procesadores que pueden o no compartir un mismo espacio de direcciones de memoria física.

Actualmente, el uso de un sólo procesador no es suficiente para cubrir las demandas computacionales de las aplicaciones modernas, que requieren en muchos casos del uso

de varios procesadores. Además, con los límites actuales del hardware, es más fácil incrementar la capacidad de cálculo con un multiprocesador que realizarlo con sistemas uniprocesadores de equivalente capacidad, y también cuesta menos. Sumado a esto, el hecho de que existen varias herramientas de diseño para este tipo de sistemas y que cada vez más crece su disponiblidad comercial, han motivado a la comunidad de tiempo real a investigar en el área de sistemas multiprocesador, en las últimas décadas. Además, los sistemas multiprocesador ofrecen soporte a otros modelos útiles para los sistemas de tiempo real, como por ejemplo, los modelos de tolerancia a fallos.

Por otra parte, en el diseño de dispositivos portables y móviles, existe un conflicto: estos sistemas deben ser diseñados tal que se maximice la vida de la batería; pero como son sistemas inteligentes, necesitan procesadores poderosos, que consumen más energía y reducen de esta forma la duración de la batería. Por esto, es crucial hallar un compromiso entre el desempeño y la duración de la batería del dispositivo. En sistemas de tiempo real, además, se deben tener en cuenta las restricciones de tiempo inherentes a este tipo de sistemas para no comprometer los cumplimientos de los plazos al disminuir el consumo de energía del sistema.

El estudio de la planificación de tiempo real es amplísimo y complejo, y existen aún muchos problemas no resueltos, algunos de ellos desde hace un año presentados en el Seminario Internacional de Problemas No Resueltos en Planificación de Tiempo Real (*International Real-Time Scheduling Open Problems Seminar (RTSOPS)*).

## 1.2. Definiciones y notaciones básicas

Un sistema de tiempo real, en general, está compuesto de un conjunto finito de tareas de tiempo real $\tau = \{\tau_1, \tau_2, \ldots, \tau_n\}$. Cada tarea (*task*) $\tau_i$ consiste en un conjunto de acciones similares que pudieran repetirse a lo largo del tiempo, y está compuesta de unidades básicas de ejecución llamadas instancias de la tarea (*jobs*), las cuales tienen restricciones temporales comunes. Las instancias de las tareas son la mínima unidad de cómputo de un sistema de tiempo real y están caracterizadas por tres parámetros:

1. Un **tiempo de activación** (*release time*) $A_i$, que es el momento en que la instancia de la tarea está lista para ejecutarse.

2. Un **tiempo de ejecución** (*execution time*) $C_i$, que es la cantidad máxima de tiempo de procesamiento que requiere la tarea para finalizar su ejecución.

3. Un **plazo** (*deadline*) $D_i$, que es la cantidad de tiempo (después del tiempo de llegada) en que la tarea debe completar su ejecución.

El sistema de tiempo real más estudiado es el **modelo de tareas periódicas** de Liu y Layland [LL73]. Una tarea periódica (*periodic task*) consiste en una sucesión periódica de instancias de tarea idénticas, las cuales están separadas por una cantidad constante de tiempo llamada **período**, que es denotada por $T_i$. Una tarea es

**esporádica** (*sporadic task*) cuando las llegadas de sus instancias al sistema están separadas por cantidades de tiempo distintas, pero siempre existe una separación mínima entre ellas. Las tareas **aperiódicas** (*aperiodic tasks*), por su parte, no tienen restricciones en los tiempos entre llegadas, y generalmente responden a eventos que ocurren de forma aleatoria. La mayoría de la investigación en la planificación multiprocesador se ha hecho sobre los modelos de tareas periódicas y los modelos de tareas esporádicas [DB09].

El sistema se denomina **síncrono** (*synchronous*) si existe algún instante de tiempo en el que todas las tareas llegan simultáneamente, y de otra manera es llamado **asíncrono** (*asynchronous*). En lo que respecta a los plazos de las tareas, existen tres niveles estudiados en la literatura:

- **Plazos implícitos** (*implicit deadlines*): todos los plazos de las tareas coinciden con los períodos ($D_i = T_i \ \forall i$).
- **Plazos restrictivos** (*constrained deadlines*): todos los plazos de las tareas deben ser no mayores que sus períodos ($D_i \leq T_i \ \forall i$).
- **Plazos arbitrarios** (*arbitrary deadlines*): no existen restricciones entre los plazos y los períodos.

A veces se le llama **peso** de una tarea a su utilización: $u_i = C_i/T_i$, que denota en qué porcentaje esta tarea hace uso del procesador. La utilización de un conjunto de tareas $\tau$ es la suma de las utilizaciones de cada tarea:

$$U(\tau) = \sum_{i=1}^{n} u_i = \sum_{i=1}^{n} \frac{C_i}{T_i}. \tag{1.1}$$

La **planificación** consiste en asignar un orden de acceso a los recursos a un conjunto de tareas que se ejecutan concurrentemente. Existen dos políticas principales de planificación que son: la **estática** y la **dinámica**. En la primera, la planificación se hace antes de ponerse el sistema en ejecución. Se deben conocer por adelantado los parámetros que caracterizan al conjunto de tareas, y típicamente se usan estimados del peor caso de los tiempos de ejecución para tomar las decisiones de planificación. Esto provoca que pueda existir una gran cantidad de tiempo de holgura, que es el tiempo en que el procesador está inactivo, pero tiene la ventaja que garantiza el cumplimiento de plazos para sistemas de tiempo real críticos. La segunda política de planificación puede usarse cuando no se necesita garantizar por adelantado el cumplimiento de plazos de las tareas. Con esta estrategia, las tareas son planificadas en tiempo de ejecución (*on-line*).

La planificación de tiempo real, además, consta de dos aspectos fundamentales: un algoritmo de planificación y una prueba de planificabilidad. El algoritmo de planificación no se restringe a asignar solamente tiempo de CPU, sino que también puede incluir criterios de asignación de recursos. Principalmente, el algoritmo de planificación se encarga de asignar un orden de acceso de las tareas a los recursos, en

particular al CPU.

Para cada algoritmo de planificación, existen una o varias pruebas de planificabilidad (*feasibility test*), las cuales son ecuaciones matemáticas que permiten verificar si bajo el algoritmo en cuestión, un conjunto determinado de tareas cumplirá o no con sus restricciones de tiempo. Sin este análisis, no se tendría ninguna garantía de que el sistema responda en los plazos asignados, lo cual es vital sobre todo para los sistemas de tiempo real críticos. Estas pruebas pueden ser condiciones suficientes o necesarias y suficientes:

- **Condición suficiente**: Si el conjunto de tareas cumple con la prueba, entonces es planificable (cumple con sus plazos). Esto no quiere decir que si no cumple con ella, no es planificable. Además, un conjunto de tareas puede satisfacer una determinada prueba de planificabilidad y no otra, dependiendo de cuán alta sea la cota que la ecuación provee.

- **Condición necesaria y suficiente**: El conjunto de tareas cumplirá sus plazos si, y sólo si, cumple con la prueba (ecuación).

Asimismo, las pruebas de planificabilidad pueden involucrar diferentes variables como el número de tareas, número de procesadores, parámetros del conjunto de tareas (tiempos de ejecución, plazos, períodos), entre otros. Mientras más parámetros involucre la prueba, tomará más tiempo realizarla, lo cual puede atentar contra el cumplimiento de plazos en un sistema donde sea necesario hacer esta prueba en línea.

Cada prueba de planificabilidad provee una cota de utilización del procesador por debajo de la cual cualquier conjunto de tareas cumplirá con sus plazos. A estas cotas se le llaman **cotas de planificabilidad**. Tomemos, como ejemplo muy simple, la prueba suficiente de planificabilidad dada por Liu y Layland para el algoritmo de tasa monótona (*Rate Monotonic*) (que será explicado más adelante):

**Teorema 1.2.1.** ***(Condición de Liu y Layland)****[LL73] Sea $n$ el número de tareas en el conjunto $\tau$. Si la utilización total $U(\tau)$ (1.1) satisface la siguiente ecuación, entonces $\tau$ será factible de planificar bajo el algoritmo Rate Monotonic:*

$$U(\tau) \leq n(2^{\frac{1}{n}} - 1). \tag{1.2}$$

*Cuando $n \to \infty$, entonces $n(2^{\frac{1}{n}}) \to ln(2)$.*

Pero esta cota de utilización es muy pesimista, puesto que impone un límite de $ln(2)$, lo cual es aproximadamente 0.693, cuando el número de tareas tiende a infinito (véase la Fig. 1.1). De hecho, ¿qué sucede si $U(\tau)$ no cumple con la ecuación? Puede que, de todas formas, el conjunto de tareas se pueda planificar. En estas pruebas de planificabilidad, es deseable que las cotas sean altas, pues si sólo se tiene una condición suficiente, mientras más alta sea la cota, es más probable que el conjunto de tareas cumpla con la ecuación de la prueba. Por esta causa, gran parte de la investigación

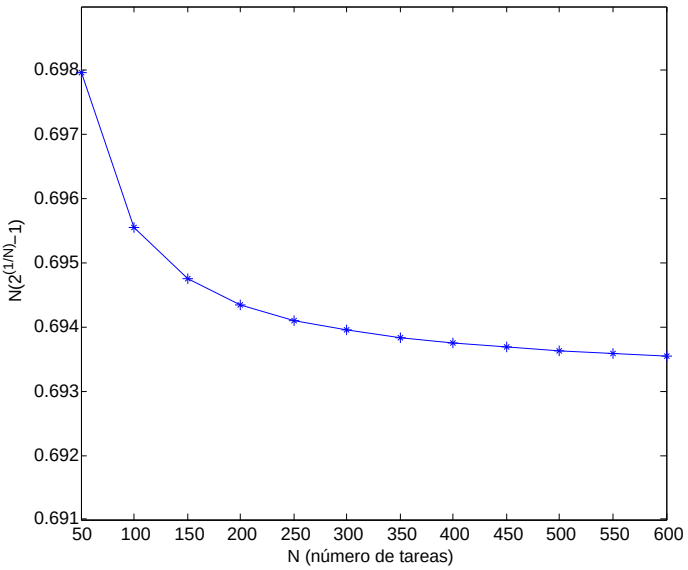


Figura 1.1: Cota de utilización para la prueba de planificabilidad de Liu y Layland.

en el campo de la planificación de tiempo real se ha invertido en subir las cotas de las pruebas de planificabilidad.

El plan de ejecución generado por un determinado algoritmo de planificación para un conjunto de tareas se denomina un **plan válido** si:

- Asigna como máximo una instancia de tarea a la vez a cada procesador.
- Asigna como máximo un procesador a la vez a cada instancia de tarea.
- No ejecuta ninguna instancia de tarea antes de su tiempo de activación.
- Asigna a cada tarea un tiempo de procesador igual a su tiempo de ejecución real o máximo.
- Satisface todas las relaciones de precedencia y todas las restricciones en el uso de los recursos.

El plan realizado por el planificador se dice que es **factible** o admisible (*feasible schedule*) si es un plan válido y asegura que todas las tareas cumplen con sus plazos.

A un conjunto de tareas $\tau$ se le llama **planificable** (*schedulable*) si existe al menos un algoritmo de planificación que pueda generar un plan admisible para $\tau$. Por otra parte, se dice que un sistema de tiempo real es planificable bajo un determinado algoritmo de planificación si este algoritmo siempre produce planes admisibles.

Respecto a los algoritmos, se dice que el algoritmo A es **óptimo** si cualquier conjunto de tareas factible de planificar, es planificable bajo el algoritmo A (siempre puede hallar un plan admisible cuando éste exista). Un problema crucial de la planificación de tiempo real es determinar si, dado un conjunto $\tau$ de tareas y un ambiente de planificación, existe un plan de ejecución para $\tau$ tal que todas las tareas cumplan sus plazos.

## 1.3. Motivación de la tesis

El problema de la planificación en un sistema de tiempo real multiprocesador puede ser formulado de varias formas y tomando en cuenta diferentes restricciones. Dos variantes importantes son la planificación en procesadores homogéneos y la planificación en procesadores heterogéneos. La primera se refiere a planificar el conjunto de tareas en $m$ procesadores de características idénticas, de forma que se maximice la cantidad de tareas que cumplen sus plazos. Por otro lado, la planificación en procesadores heterogéneos no requiere que los procesadores sean idénticos, y por ende, las tareas podrán tener un tiempo de ejecución diferente dependiendo del procesador en que se ejecuten.

Otra de las variantes de la planificación multiprocesador es asumir que se tiene un número fijo de procesadores, ó, por el contrario, que el número de procesadores es infinito. Estas son dos estrategias que han sido utilizadas para medir el desempeño de los algoritmos de planificación. El primer caso tiene el propósito de verificar si el algoritmo planifica correctamente el conjunto de tareas en los $m$ procesadores disponibles, ó que el mayor número de tareas cumplan con sus plazos de manera tal que se maximice la utilización de cada procesador, en caso que los plazos no sean críticos. En el segundo caso, se busca minimizar el número de procesadores que se necesita para planificar la totalidad de las tareas bajo el algoritmo. Mientras menor sea el número de procesadores a utilizar, mejor será el algoritmo.

Comparada con la planificación en un sistema uniprocesador, la planificación en sistemas multiprocesador es mucho más compleja. Determinar un plan de ejecución óptimo para un conjunto independiente de tareas en un procesador tiene complejidad polinomial en el número de tareas; pero hacer esto para varios procesadores se ha demostrado que es un problema NP-completo, y por lo tanto de los más complejos que existen [GJ75].

La planificación en sistemas multiprocesador consiste en dos subproblemas: (1) la asignación de las tareas a los procesadores y (2) la planificación de las tareas en los procesadores individuales. El problema de la asignación está relacionado con cómo hacer una partición del conjunto de tareas para ser asignadas a los distintos procesadores. Se ha demostrado que en el esquema particionado (que es uno de los dos esquemas de la planificación multiprocesador, y será explicado más adelante), el problema de la asignación, que consiste en asignar las tareas a los procesadores, es *NP-*

*hard* [LW82]. Por lo tanto, determinar una solución óptima tiene complejidad exponencial, lo cual repercute en los tiempos de respuesta del sistema. Por esta razón, la mayoría de los algoritmos son en la práctica heurísticos. Luego de hacer una asignación de tareas a los procesadores, la segunda fase de la planificación se reduce a planificar exitosamente cada conjunto de tareas en cada procesador de forma individual, siguiendo un determinado algoritmo de planificación uniprocesador y alguna prueba de factibilidad.

Desafortunadamente, algunos de los problemas relacionados con la planificación de tiempo real para múltiples procesadores aún no tienen solución conocida, o han sido resueltos sólo parcialmente. Muchos de los resultados conocidos concernientes a la planificación en un sólo procesador no pueden ser extendidos al caso de varios procesadores. Además, el análisis de planificación de los algoritmos existentes para múltiples procesadores, ofrece cotas muy pesimistas. Por esta razón, se hace importante encontrar nuevos algoritmos que ofrezcan cotas más altas de planificación para, de esta forma, tener mayor garantía en lo que respecta al cumplimiento de plazos.

El problema de la asignación ha sido ampliamente estudiado en la literatura, y ha sido extendido a incluir procesadores con diferentes características, y varios modelos de tareas. Sin embargo, la dirección principal de la investigación ha sido encontrar soluciones aproximadas en tiempo polinomial por medio de heurísticas, dada la condición *NP-hard* del problema. Ha habido muy pocos trabajos que se enfoquen en aplicar técnicas exactas de optimización para obtener la planificación óptima en sistemas multiprocesador de tiempo real.

En la práctica, existen muchos sistemas embebidos y de tiempo real que no son de tamaño grande, debido a requerimientos típicos de costo, y restricciones de espacio y energía en la plataforma de hardware. Para estos casos, pudiera ser adecuado utilizar un algoritmo exacto para hallar la asignación óptima, pues éstos siempre ofrecerán mejores soluciones que los algoritmos aproximados. Entiéndase por algoritmo exacto, uno que no es heurístico, que en nuestro caso particular, asignará todas las tareas en caso que esto sea posible, y si no las asigna todas, es porque esto no es posible de hacer. Por el contrario, un algoritmo heurístico muchas veces asigna menos tareas de las que es posible asignar, aunque tenga baja complejidad. Un algoritmo exacto pudiera obtener mejores soluciones que uno aproximado en tiempos de cómputo aceptables, teniendo en cuenta la naturaleza del problema que se está enfrentando.

En este contexto, esta tesis usa el mismo enfoque del trabajo hecho en 2008 por Baruah y Bini [BB08]. Su investigación modela el problema de la asignación como un problema de programación lineal entera con variables binarias. Sin embargo, no se aplica ningún método para obtener una solución. Por lo tanto, su trabajo no arroja ninguna luz sobre la factibilidad de aplicar esta técnica en la práctica, ni tampoco se da ningún indicio de qué tipo de implementación real pudiera ser útil para obtener resultados experimentales significativos.

Esta existencia de algunos trabajos previos enfocados en hallar una solución óptima para la planificación de un conjunto de tareas bajo el esquema particionado, pero sin resultados experimentales disponibles para el modelo de tareas que aquí se propondrá, nos mueve a buscar técnicas de optimización para evaluar la factibilidad de aplicarlas, con el objetivo de resolver este problema particular.

## 1.4. Objetivos

Este trabajo parte de la siguiente hipótesis: para problemas de tamaño pequeño propio de muchos sistemas reales, en la práctica, puede ser una opción factible utilizar algoritmos exactos para encontrar una asignación óptima fuera de línea de un conjunto de tareas en un sistema multiprocesador.

El objetivo de este proyecto de tesis es evaluar, para problemas prácticos, la factibilidad de la aplicación de técnicas conocidas de optimización entera al problema de la asignación en un sistema multiprocesador.

Para conseguir nuestro objetivo, se aborda el problema de la siguiente forma:

- Se establece el modelo de tareas sobre el cual se trabajará.
- Se modela matemáticamente el problema de la asignación en un sistema multiprocesador de tiempo real como un problema de optimización entera binaria, bajo el modelo de tareas establecido.
- Se propone la utilización de un algoritmo conocido del área de investigación de operaciones para obtener una solución óptima del problema atacado.
- Se adapta el algoritmo original propuesto al problema de la asignación, tomando en cuenta las particularidades de este problema.
- Se evalúa el desempeño del algoritmo (implementado en lenguaje C) mediante resultados experimentales realizados a ciertos conjuntos de tareas con características típicas, para obtener los tiempos de respuesta y las tasas de aceptación.

## 1.5. Estructura de la tesis

El Capítulo 1 está dedicado a una breve introducción a los sistemas de tiempo real. Además, incluye una motivación a este trabajo de tesis y algunas definiciones y notaciones básicas para la comprensión de la problemática abordada.

En el Capítulo 2 se expone una revisión del estado del arte de la investigación relacionada con la planificación multiprocesador en sistemas de tiempo real, haciendo un resumen sobre los resultados más importantes que existen en el área. Igualmente, se exponen trabajos relacionados con el objetivo de esta tesis.

En el Capítulo 3 se presenta un breve estudio de Programación Lineal y más específicamente, de Programación Lineal Entera Binaria, necesario para la resolución del problema que será formulado.

En el Capítulo 4 se presenta el algoritmo propuesto para resolver el modelo de optimización que modela el problema de la planificación de un conjunto de $m$ tareas en $n$ procesadores, bajo el esquema particionado. De igual forma, se presenta el modelo de tareas escogido bajo el cual se modeló el problema.

En el Capítulo 5, se presentan los resultados experimentales del método propuesto (implementado en lenguaje C) del algoritmo, evaluado en ciertos conjuntos de tareas, con el objetivo de evaluar su desempeño.

Finalmente, se presentan las conclusiones de la tesis y las posibles direcciones de trabajo futuro en el Capítulo 6.

# Capítulo 2

# Estado del arte

Muchos autores han investigado en el tema de la planificación de tiempo real para múltiples procesadores, proponiendo algoritmos y estableciendo teoremas que dan condiciones suficientes ó necesarias y suficientes de planificabilidad. En las siguientes secciones se expondrán algunos de los resultados más importantes en el área hasta la actualidad.

## 2.1. Planificación uniprocesador

Esta sección describe brevemente algunos resultados importantes en la planificación uniprocesador, relevantes al estudio de la planificación multiprocesador.

### 2.1.1. Planificación de prioridades estáticas

El algoritmo de planificación de prioridades estáticas más conocido en los sistemas uniprocesador de tiempo real es *rate-monotonic* (RM). Es uno de los primeros algoritmos de planificación uniprocesador de tiempo real y fue desarrollado por Liu y Layland en 1973 [LL73].

En los algoritmos de prioridades estáticas, a cada tarea y a cada instancia de las tareas se les asigna una prioridad que nunca cambia. En cada instante de planificación, se ejecuta la instancia de tarea que tenga la mayor prioridad, expropiando la tarea que se esté ejecutando en ese momento, si tuviese una menor prioridad.

El algoritmo RM asigna las prioridades a las tareas de acuerdo a sus períodos: mientras más corto sea el período de la tarea, mayor será su prioridad. Las prioridades son estáticas porque no varían, pues los períodos de las tareas no cambian con el tiempo. RM sólo puede ser aplicado a sistemas de tareas periódicas cuyos plazos coincidan con los períodos. Existen otros algoritmos óptimos para modelos más generales, como el DM (*Deadline Monotonic*) [LW82], pero no serán tratados aquí.

En [LL73], Liu y Layland demostraron que el algoritmo *rate-monotonic* es óptimo:

si un sistema de tareas periódicas es factible de planificar por algún algoritmo de planificación de prioridades estáticas, entonces el sistema de tareas es planificable bajo RM. En el mismo trabajo, se ofrece una condición suficiente de planificabilidad para un sistema periódico de tareas en un ambiente uniprocesador. Esta prueba sólo involucra el número de tareas, pero proporciona una cota muy baja, y es la que se dio en el Teorema 1.2.1 en el Capítulo 1. Más tarde, en [JPLD89] se muestra una condición necesaria y suficiente de planificabilidad para RM (Test exacto de Lehoczky), que tiene una alta complejidad. Se requiere comprobar la Ecuación 2.1 para todos los puntos de planificación en $S_i$, cuya cardinalidad es una función de los valores relativos de los períodos de las tareas, por lo que puede ser verificada en tiempo pseudo-polinomial. Debido a su alta complejidad, esta prueba no es muy usada en la práctica, aunque proporciona una prueba exacta:

**Teorema 2.1.1.** *[JPLD89] Sea $\tau = \{\tau_1, \ldots, \tau_n\}$ un sistema de tareas periódicas tal que: $T_1 \leq T_2 \leq \ldots \leq T_n$. Sea además, $S_i = \{kT_j \mid j = 1, \ldots, i; k = 1, \ldots, \lfloor T_i/T_j \rfloor\}$ y $W_i(t) = \sum_{j=1}^{i} e_j \lfloor t/T_j \rfloor$. Entonces, $\tau_i$ es planificable bajo RM si, y sólo si:*

$$L_i = \min_{t \in S_i}(W_i(t)/t) \leq 1. \tag{2.1}$$

*El conjunto completo de tareas $\tau$ es planificable bajo RM si, y sólo si:*

$$L = \max_{1 \leq i \leq n} L_i \leq 1. \tag{2.2}$$

Fue también Lehoczky quien condujo un estudio estadístico que demostró que, para conjuntos de tareas aleatoriamente generados, la utilización promedio realmente planificable de RM es aproximadamente del 88 %, muy por encima de la cota de Liu y Layland ($\approx 69.3\,\%$) [JPLD89].

Las pruebas suficientes de planificabilidad más usadas para el caso uniprocesador bajo RM, además de la condición de Liu y Layland son las que se citan a continuación:

**Teorema 2.1.2.** ***Condición de Período Creciente (IP):****[DL78] Sea $\tau = \{\tau_1, \ldots, \tau_n\}$ un conjunto de tareas con $T_1 \leq T_2 \leq \ldots \leq T_n$ y sea:*

$$U_{n-1} = \sum_{i=1}^{n-1} \frac{C_i}{T_i} \leq (n-1)(2^{1/(n-1)} - 1). \tag{2.3}$$

*Si la siguiente condición se cumple:*

$$u_n \leq 2\left(1 - \frac{U_{n-1}}{n-1}\right)^{-(n-1)} - 1, \tag{2.4}$$

*entonces, el conjunto de tareas tendrá una planificación factible bajo el algoritmo RM. Cuando $n \to \infty$, la mínima utilización de la tarea $\tau_n$ tiende a $(2e^{-u} - 1)$.*

**Teorema 2.1.3.** ***Condición orientada a la utilización (UO):*** *[OS95] Sea* $\tau = \{\tau_1, \ldots, \tau_{n-1}\}$ *un conjunto de* $(n-1)$ *tareas planificables bajo RM. Una nueva tarea* $\tau_n$ *puede ser planificable junto con las* $(n-1)$ *anteriores en el sistema, si la siguiente condición se satisface:*

$$\frac{C_n}{T_n} \leq 2\left(\prod_{i=1}^{n-1}(1+u_i)\right)^{-1} - 1. \tag{2.5}$$

**Teorema 2.1.4.** ***Condición orientada al período (PO):*** *[ABS95] Dado un conjunto de tareas* $\tau = \{\tau_1, \ldots, \tau_n\}$*, defínanse* $S_i$ *y* $\beta$ *como sigue:*

$$S_i = \log_2 T_i - \lfloor \log_2 T_i \rfloor, \;\; i = 1, \ldots, n. \tag{2.6}$$

$$\beta = \max S_i - \min S_i, \;\; i = 1, \ldots, n. \tag{2.7}$$

*Si la utilización total satisface la siguiente ecuación:*

$$U(\tau) \leq \max\{\ln 2, 1 - \beta \ln 2\}, \tag{2.8}$$

*entonces el conjunto de tareas es planificable en un procesador bajo RM.*

**Teorema 2.1.5.** ***Condición de planificabilidad RBound:*** *[SLM03] Considérese un conjunto de tareas periódicas* $\tau$*, y sea* $\hat{\tau}$ *el conjunto transformado después de ejecutar el algoritmo ScaleTaskSet*[1] *en* $\tau$*. Sea además* $r = \hat{T}_n/\hat{T}_1$*. Si:*

$$\sum_{i=1}^{n} \frac{C_i}{T_i} \leq (n-1)(r^{1/(n-1)} - 1) + \frac{2}{r} - 1, \tag{2.9}$$

*entonces, el conjunto de tareas* $\tau$ *es planificable en un procesador bajo RM.*

La existencia de la prueba exacta de Lehoczky, combinada con el hecho de que RM es un algoritmo óptimo de prioridades estáticas en un ambiente uniprocesador, implica que el análisis de planificabilidad para sistemas de tareas periódicas síncronas con plazos iguales a los períodos puede ser resuelto en tiempo pseudo-polinomial en un procesador bajo asignación estática de prioridades.

### 2.1.2. Planificación de prioridades dinámicas

Los algoritmos de planificación de prioridades dinámicas son aquellos en los que, en instantes de tiempo diferentes, una misma tarea puede tener distintas prioridades. Es decir, durante su tiempo de vida, la prioridad de una tarea puede cambiar, incluso varias veces. Para sistemas de tareas periódicas, esto significa que a las instancias de una misma tarea se les pueden asignar diferentes prioridades en el curso de la ejecución de la tarea.

[1]Este algoritmo transforma el conjunto original de tareas en un conjunto equivalente donde la razón entre el mayor y el menor período es menor que 2: $r = T_{max}/T_{min} \leq 2$ [SLM03].

El algoritmo *earliest deadline first* (EDF) es de prioridades dinámicas, y también es de los primeros algoritmos que surgieron en la planificación uniprocesador. También fue desarrollado por Liu y Layland en su artículo semilla de 1973 [LL73]. En cada instante en que se debe tomar una decisión de planificación (una tarea termina su ejecución, llega una nueva tarea al sistema, cuando se bloquea una tarea, etc.), EDF le da la mayor prioridad a la tarea cuyo plazo esté más próximo a cumplirse.

En el mismo artículo de Liu y Layland donde se probó la optimalidad de RM para un ambiente uniprocesador bajo asignación de prioridades estáticas, se demostró que EDF es óptimo bajo asignación de prioridades dinámicas para sistemas uniprocesador. Esto es, cualquier sistema de tareas periódicas síncronas cuyos plazos coincidan con los períodos que sea planificable en un procesador por algún algoritmo de planificación de prioridades dinámicas, será también planificable bajo EDF.

En lo que respecta a la política de planificación EDF para uniprocesadores, existe una prueba de planificabilidad necesaria y suficiente dada por Liu y Layland:

**Teorema 2.1.6.** *[LL73] Un sistema de tareas periódicas $\tau = \{\tau_1, \ldots, \tau_n\}$ es planificable bajo EDF si, y sólo si,*

$$\sum_{i=1}^{n} u_i \leq 1. \tag{2.10}$$

Gracias a esta condición, el problema de la planificación uniprocesador de un conjunto de tareas periódicas síncronas cuyos plazos coincidan con sus períodos puede ser resuelto en *O(n)* [Liu00].

EDF es un algoritmo de prioridades dinámicas a nivel de instancias de tareas (*job-level dynamic-priority*), porque la prioridad de una instancia de tarea permanece fija durante su tiempo de vida. Existen otros algoritmos, de prioridades plenamente dinámicas (*fully-dinamic priority*) [SBV96], en los que la prioridad de una instancia de tarea (relativa a otras instancias) puede cambiar en cualquier momento. Este tipo de algoritmos sufren de una gran sobrecarga debido a los muy frecuentes cambios de contexto, pero son interés teórico pues actualmente son el único método conocido para alcanzar la optimalidad en lo que respecta a la planificación de tareas periódicas en sistemas multiprocesador.

## 2.2. Planificación multiprocesador

En la planificación multiprocesador se deben resolver dos problemas importantes:

1. El problema de la asignación (*allocation problem*): decidir en cuál procesador se ejecutará cada tarea.

2. El problema de la planificación (*scheduling problem*): decidir cuándo y en qué orden se debe ejecutar cada instancia de tarea, respecto a otras instancias de otras tareas.

Los algoritmos de planificación multiprocesador pueden ser clasificados de acuerdo a cuándo se pueden hacer cambios de prioridad y de asignación. La clasificación basada en prioridades y basada en migraciones de [JCB04] es la siguiente:

**Asignaciones:**

1. **Sin migración:** cada tarea se asigna a un procesador y todas sus instancias siempre se ejecutan en este procesador.
2. **Migración a nivel de tareas:** Las instancias de una tarea pueden ejecutarse en diferentes procesadores, pero cada instancia de tarea sólo puede ejecutarse en un procesador.
3. **Migración a nivel de instancias de tareas:** Una instancia de tarea puede migrar y ejecutarse en diferentes procesadores en cualquier momento[2].

**Prioridades:**

1. **Prioridades fijas a nivel de tareas:** Cada tarea tiene una única prioridad fija, aplicada a cada una de sus instancias.
2. **Prioridades fijas a nivel de instancias de tareas:** Las instancias de una tarea pueden tener distintas prioridades. Sin embargo, cada instancia de tarea tiene una única prioridad estática. Un ejemplo es EDF.
3. **Prioridades dinámicas:** Una misma instancia de tarea puede tener diferentes prioridades en diferentes momentos (lo que se llamó anteriormente prioridad plenamente dinámica).

Un conocimiento *a priori* de los parámetros que caracterizan el conjunto de tareas ha conllevado a estudios fuera de línea, según se puede apreciar luego de que fue establecido el siguiente teorema:

**Teorema 2.2.1.** *[DM89] Para cualquier sistema multiprocesador, ningún algoritmo de planificación basado en plazos puede ser óptimo sin un conocimiento completo a priori de los momentos de activación, tiempos de cómputo y los plazos de las tareas.*

Este teorema plantea que es necesario el conocimiento previo de los tiempos de activación para alcanzar la optimalidad. Los algoritmos clásicos de planificación no pueden ser óptimos si se usan en línea, puesto que se requiere del conocimiento de los momentos de activación de las tareas, y no se puede esperar encontrar un algoritmo óptimo para el caso general.

### 2.2.1. Métricas de desempeño

Generalmente, han sido utilizadas cuatro métricas de desempeño para medir la efectividad de los algoritmos de planificación multiprocesador, las cuales serán explicadas en esta sección.

[2]Es válido aclarar que nunca está permitida la ejecución paralela de ninguna instancia de tarea.

#### 2.2.1.1. Cotas de utilización

Esta es una métrica muy útil para conjuntos de tareas cuyos plazos coinciden con los períodos, pero generalmente las cotas de utilización son difíciles de encontrar. La cota de utilización en el peor caso $U_A$ para un determinado algoritmo de planificación $A$ se define como la utilización máxima que puede llegar a tener un conjunto de tareas para que se garantice que es planificable bajo el algoritmo $A$. Si la utilización total del conjunto es mayor que $U_A$, ya no se garantiza el cumplimiento de plazos (aunque puede suceder que se sobrepase la cota máxima de utilización y a pesar de eso, el conjunto de tareas sea planificable bajo el algoritmo $A$). La cota $U_A$ puede ser usada como una prueba suficiente, pero no necesaria, de planificación.

#### 2.2.1.2. Medidas empíricas

Otra medida comparativa de la efectividad de un algoritmo de planificación se obtiene evaluando el número de conjuntos de tareas aleatoriamente generados que el algoritmo logra planificar. Idealmente, se compararía la cantidad de conjuntos planificables según una determinada prueba de planificación contra la cantidad de conjuntos generados que son realmente planificables. Pero para esto, se necesita tener un mecanismo que permita saber si un conjunto de tareas es planificable o no bajo determinado algoritmo. Hasta el momento no existen pruebas exactas de planificación para el caso de tareas esporádicas y son potencialmente intratables para el caso de tareas periódicas. Es por esto que típicamente se ha usado esta medida para comparar el desempeño relativo de dos o más algoritmos de planificación, o de dos o más pruebas de planificabilidad. Si se usa esta medida de comparación, es necesario usar un algoritmo de generación de tareas que sea insesgado [BB05].

#### 2.2.1.3. Razón de aproximación

Esta es una forma de comparar el desempeño de un algoritmo $A$ con el de un algoritmo óptimo $O$. Por ejemplo, considérese el problema de determinar el número mínimo de procesadores requeridos para planificar un conjunto de tareas $\tau$. Sea $M_O(\tau)$ el número de procesadores que el algoritmo óptimo $O$ requiere para planificar al conjunto $\tau$, y $M_A(\tau)$ el número de procesadores que requiere el algoritmo $A$. La razón de aproximación $\mathcal{R}_A$ del algoritmo $A$ está dada por:

$$\mathcal{R}_A = \lim_{M_O(\tau)\to\infty} \left( \max_{\forall\tau} \frac{M_A(\tau)}{M_O(\tau)} \right)$$

El valor $\mathcal{R}_A$ es siempre mayor o igual a uno. Mientras más se acerque a 1, mejor será el algoritmo $A$ de planificación. $\mathcal{R}_A = 1$ significa que el algoritmo $A$ es óptimo.

#### 2.2.1.4. Aumento de recursos

El factor de aumento de recursos [KP95] también es un método alternativo para comparar el desempeño de un algoritmo $A$ de planificación con el de un algoritmo

óptimo $O$. En vez de considerar cuántos procesadores más que $O$ necesitaría $A$ para planificar un conjunto de tareas, se considera qué incremento en la velocidad del procesador se requeriría, asumiendo que los tiempos de ejecución de las tareas decrecen linealmente con respecto a la velocidad de procesamiento.

El factor de aumento de recursos $f$ para un algoritmo de planificación $A$ se define como el factor mínimo por el cual la velocidad de los $m$ procesadores tendría que ser aumentada para que todos los conjuntos de tareas que son planificables bajo un algoritmo óptimo en $m$ procesadores de velocidad 1, se vuelvan planificables bajo el algoritmo $A$.

Sea $\tau$ un conjunto de tareas que es factible de planificar en un sistema de $m$ procesadores de velocidad unitaria de procesamiento. Asumamos que usando el algoritmo $A$, el mismo conjunto $\tau$ es planificable en un sistema de $m$ procesadores, cada uno de velocidad $f(\tau)$. El *factor de aumento de recursos* $f_A$ para el algoritmo $A$ está dado por:

$$f_A = \max_{\forall m, \forall \tau}(f(\tau))$$

De igual manera que en la métrica anterior, $f_A \geq 1$, y mientras más pequeño sea el valor, más efectivo será el algoritmo. $f_A = 1$ significa que el algoritmo $A$ es óptimo.

### 2.2.2. Comparabilidad

Cuando se quieren comparar dos algoritmos de planificación multiprocesador $A$ y $B$, se tienen tres posibilidades:

1. **Dominancia:** El algoritmo $A$ domina al algoritmo $B$ si todos los conjuntos de tareas que son planificables bajo $B$ también lo son bajo $A$, y existen conjuntos que son planificables bajo $A$, pero no lo son bajo $B$.

2. **Equivalencia:** Los algoritmos $A$ y $B$ son equivalentes, si todos los conjuntos de tareas que son planificables bajo el algoritmo $B$ también lo son bajo el algoritmo $A$ y viceversa.

3. **Incomparabilidad:** Los algoritmos $A$ y $B$ son incomparables si existen conjuntos de tareas que son planificables bajo el algoritmo $A$, pero no bajo el algoritmo $B$, y viceversa.

En [JCB04] se considera la relación entre las nueve clases diferentes de algoritmos de planificación multiprocesador, que resultan de las combinaciones de las tres clases de migraciones y las tres categorías de prioridades descritas en la Sección 2.2. Los resultados principales son los siguientes:

- La planificación global de prioridades dinámicas (con migración a nivel de instancias de tareas) domina todas las otras clases.

- Las tres clases con prioridades fijas (esquema particionado, con migración a nivel de tareas y con migración a nivel de instancias de tareas) son incomparables.

- Las tres clases particionadas (con prioridades fijas a nivel de tarea, prioridades fijas a nivel de instancias de tareas y prioridades dinámicas) son incomparables respecto a las tres clases con migración a nivel de tarea.

### 2.2.3. Cotas para la planificación multiprocesador

Es deseable que existan condiciones necesarias y suficientes para la planificabilidad de un conjunto de tareas en el caso multiprocesador, como existen condiciones concretas para el caso uniprocesador, pero hasta el momento, todavía no las hay. Un resultado importante que provee una condición suficiente de planificabilidad, también dado en [DM89], es el siguiente:

**Teorema 2.2.2.** *[DM89] Para un conjunto de tareas periódicas independientes cuyos plazos coincidan con sus períodos y que respeten la condición necesaria de planificabilidad $U(\tau) \leq m$, existe una planificación factible si $D\frac{C_i}{D_i} \in \mathbb{N}$, para todo $i = 1, \ldots, n$, donde $D = gcd(D_1, \ldots, D_n)$.*

Los siguientes teoremas muestran la dificultad para encontrar una planificación no expropiativa para un conjunto de tareas independientes con el mismo plazo:

**Teorema 2.2.3.** *[GJ75] Considérese el problema de la planificación multiprocesador no expropiativa con 2 procesadores y sin compartición de recursos, para tareas independientes. Para tiempos de cómputo arbitrarios ó 1 o 2 unidades de tiempo, este problema es $NP$-completo.*

**Teorema 2.2.4.** *[GJ75] El problema de la planificación multiprocesador no expropiativa con al menos 3 procesadores, y un recurso compartido, para tareas independientes con un mismo tiempo de ejecución es $NP$-completo.*

En la planificación multiprocesador, existen dos esquemas para la distribución de las tareas entre los procesadores: el esquema global y el particionado, que serán explicados en las próximas dos secciones. Un resultado importante es que estos dos enfoques no son comparables bajo una planificación estática, en el sentido que bajo algoritmos de asignación estática de prioridades, existen conjuntos de tareas que son planificables usando el esquema particionado y no lo son usando el esquema global, y viceversa [LW82].

En [BAJ03][SB02] se expone otro importante resultado: una cota superior para la utilización planificable de cualquier algoritmo de planificación multiprocesador de prioridades estáticas o de prioridades dinámicas (a nivel de instancias de tareas) en cualquiera de los dos esquemas, particionado o global.

**Teorema 2.2.5.** *[BAJ03, SB02] Ningún algoritmo de planificación multiprocesador de prioridades estáticas o de prioridades dinámicas a nivel de instancias de tareas (esquema particionado o global) puede planificar todos los conjuntos de tareas periódicas con utilización a lo sumo $U$ en $m$ procesadores si $U > (m+1)/2$.*

Nótese que esto se cumple para RM y EDF, pues el primero es un algoritmo de prioridades estáticas y el segundo es un algoritmo de prioridades dinámicas a nivel de instancias de tareas. Por lo tanto, a menos que se utilice un algoritmo plenamente dinámico, la máxima utilización que se puede esperar es muy cercana al 50 %.

### 2.2.4. Esquema particionado

En este esquema (Figura 2.1), existe un control de admisión que admite o no las tareas en el sistema, y las asigna a los procesadores. Una vez que una tarea es asignada a un procesador, pasa a formar parte de su cola local y todas las instancias de ella que se activan periódicamente, se ejecutan en ese mismo procesador, bajo su planificador local.

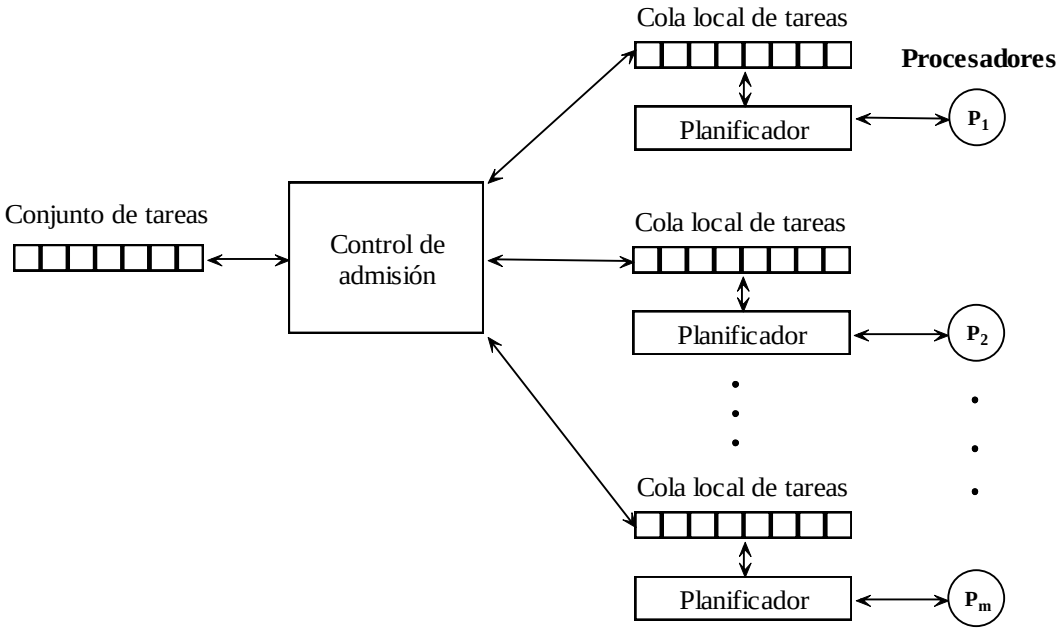


Figura 2.1: Esquema particionado.

El enfoque más usado para definir un algoritmo de planificación multiprocesador para el esquema particionado, consiste en escoger:

1. Una heurística para resolver el problema de la asignación.

2. Un algoritmo de planificación uniprocesador para planificar los conjuntos de tareas en cada procesador.

3. Una condición de planificabilidad asociada con el algoritmo de planificación para verificar el cumplimiento de plazos.

Cuando surgió la planificación multiprocesador, el esquema particionado fue mucho más estudiado que el global, pues tiene la ventaja que el problema de la planificación se reduce a la planificación en un solo procesador (para cada uno de ellos), luego de que las tareas hayan sido asignadas a los procesadores. Por lo tanto, para resolver este problema, se puede escoger alguno de los muchos algoritmos propuestos para la planificación en sistemas uniprocesador, para los cuales existen pruebas eficientes de planificabilidad. Además, como el particionado (asignación) de las tareas se realiza *a priori*, no se incurre en sobrecarga en tiempo de corrida como ocurre con el esquema global. En la actualidad, el estado del arte parece favorecer este esquema [BB08].

El problema de la asignación es un problema de mochila múltiple, que es *NP-hard* [HK04]. En un problema de mochila unidimensional (*knapsack problem*) se tienen $n$ objetos con pesos $p_i$ y beneficios $b_i$, y una mochila con capacidad $C$. Imagínese un mercader que ha realizado compras en ultramar, y quiere cargar su barco de capacidad $C$ con la mayor cantidad posible de mercancía, donde cada mercancía tiene un determinado peso y ofrece una determinada ganancia (beneficio).

El objetivo en el problema de la mochila consiste en maximizar la ganancia que se obtendrá, de forma tal que los pesos de los objetos seleccionados no excedan la capacidad de la mochila. El problema de mochila unidimensional se formula de la siguiente forma, donde cada variable $X_i$ tomará el valor de uno si se coloca el objeto en la mochila y cero si no se coloca: [HK04]

$$\text{máx} \sum_{j=1}^{n} b_j X_j \tag{2.11}$$

$$s.a. \sum_{j=1}^{n} p_j X_j \leq C, \tag{2.11a}$$

$$X_j \in \{0,1\}, j = 1, \ldots, n. \tag{2.11b}$$

Cuando se tiene más de un recipiente, se trata de un problema de mochila múltiple (*multiple knapsack*, también conocido como *bin-packing*). En este caso, existen $n$ objetos a ser colocados en $m$ recipientes, de forma tal que se maximice la ganancia total y, al mismo tiempo, no se exceda la capacidad $c_i$ de ningún recipiente, y en donde cada objeto sea colocado a lo sumo en un recipiente:

$$\text{máx} \sum_{i=1}^{m} \sum_{j=1}^{n} b_i X_{ij} \tag{2.12}$$

$$s.a. \sum_{j=1}^{n} p_j X_{ij} \leq c_i,\ i = 1, \ldots, m, \tag{2.12a}$$

$$\sum_{i=1}^{m} X_{ij} \leq 1,\ j = 1, \ldots, n, \tag{2.12b}$$

$$X_{ij} \in \{0, 1\},\ i = 1, \ldots, m,\ j = 1, \ldots, n, \tag{2.12c}$$

$$X_{ij} = \begin{cases} 1 \text{ si el objeto } i \text{ -ésimo se asigna al recipiente } j \text{ -ésimo,} \\ 0 \text{ en caso contrario.} \end{cases} \tag{2.12d}$$

El problema de la asignación en la planificación multiprocesador ha sido analizado mayormente con heurísticas, ya que hasta el momento no se conoce ningún algoritmo para resolverlo en tiempo polinomial. De hecho, dicho algoritmo no existirá, a menos que $P = NP$. Las heurísticas más conocidas en la literatura para resolver el problema de *bin-packing* son (véase [ECV98]):

- *First-Fit* (FF): Considerando que los recipientes están numerados, este algoritmo asigna el próximo objeto al recipiente no vacío con menor índice, tal que la suma del peso del nuevo objeto y el peso total de los objetos ya contenidos en el recipiente no exceda su capacidad. Si el nuevo objeto excede la capacidad de todos los recipientes no vacíos, entonces el algoritmo lo asigna a un recipiente vacío.

- *Best-Fit* (BF): Este algoritmo trata de asignar el próximo objeto al recipiente no vacío con la menor capacidad disponible. Si hay más de un recipiente con la misma capacidad disponible, entonces BF asigna el objeto al recipiente de menor índice. Si el nuevo objeto excede la capacidad de todos los recipientes no vacíos, BF lo asigna a un nuevo recipiente vacío.

- *Next-Fit* (NF): Después de asignar el primer objeto al primer recipiente, NF asigna el próximo objeto al mismo recipiente solamente si su peso no excede la capacidad del recipiente. De lo contrario, NF asigna el objeto al próximo recipiente, que está vacío. Una desventaja de este algoritmo es que no verifica si el objeto puede asignarse a recipientes anteriores al que se está manejando, por lo que puede que se desperdicie mucha capacidad en esos recipientes previos.

- *Worst-Fit* (WF): Este algoritmo es similar a BF, pero con la diferencia que éste asigna un objeto al recipiente con la mayor capacidad disponible tal que el peso del objeto no exceda la capacidad del recipiente.

Existen también versiones de estas heurísticas en el campo de la planificación multiprocesador, como son BFD y FFD (*Best Fit Decreasing, First Fit Decreasing*), donde

las tareas se ordenan en orden no creciente de sus utilizaciones, antes de aplicar la heurística correspondiente. Similarmente, una I en lugar de la D denota que el ordenamiento se realiza en orden no decreciente de sus utilizaciones.

En la literatura existen diversos trabajos de distinta naturaleza, donde varían los modelos de hardware y software, el conjunto de restricciones y la estrategia de solución. Entre éstas, podemos encontrar, entre otras: algoritmos genéticos [KD08, POY07], búsqueda Tabú, redes neuronales y programación con restricciones [PEHJ08].

Aunque los algoritmos genéticos son una de las heurísticas más conocidas actualmente para resolver problemas complejos, y de hecho han sido aplicados a la planificación multiprocesador de tiempo real [KD08, IASAS08], para problemas clasificados $NP-hard$, no ofrecen mejor desempeño que un algoritmo que haya sido especialmente diseñado para un problema particular [POY07]. Para este tipo de problemas, los algoritmos genéticos están enfocados en obtener soluciones aproximadas en tiempo polinomial. Es por esto que, aunque se haya demostrado que en la práctica tienen un buen desempeño, esta técnica no es apropiada para nuestro propósito, que es obtener una solución óptima.

#### 2.2.4.1. Esquema particionado con prioridades estáticas

Varios algoritmos para la planificación multiprocesador han sido definidos, fijando la heurística deseada para la asignación de tareas a los procesadores. Por ejemplo, considérese el algoritmo de planificación que usa la heurística FF de *bin-packing* para realizar la asignación de las tareas a los procesadores, el algoritmo RM para planificar localmente las tareas en cada procesador y la cota de utilización de Liu y Layland (Teorema 1.2.1) como prueba de planificabilidad, y llamémosle a este algoritmo RM-FF. Análogamente, se pueden definir los algoritmos RM-BF (RM y *Best-Fit*), RM-NF (RM y *Next-Fit*) y RM-WF (RM y *Worst-Fit*). También existen algunas variaciones que son, entre otras: RM-FFI (RM y *First-Fit-Increasing*) y RM-FFD (RM y *First-Fit-Decreasing*).

Oh y Baker investigaron el algoritmo RM-FF y demostraron que existe una cota superior para la utilización planificable de un conjunto de tareas, que es aproximadamente de $0.414m$[3]:

**Teorema 2.2.6.** *[OB98] Un conjunto $\tau$ de tareas periódicas es planificable bajo RM-FF en $m \geq 2$ procesadores si:*

$$U(\tau) \leq m(2^{\frac{1}{2}} - 1) \approx 0.414m. \tag{2.13}$$

También demostraron la existencia de una cota superior ligeramente mejor que $(m + 1)/2$ dada en el Teorema 2.2.5 para el caso de la planificación multiprocesador con prioridades estáticas bajo el esquema particionado (pero no el global). Este resultado

[3]En lo que sigue, se considerará que $m$ es el número de procesadores.

provee una cota superior de utilización de $(m+1)/(1+2^{1/(m+1)})$ para cualquier algoritmo de prioridades fijas (bajo el esquema particionado). Luego, en [JMLG04b] se generalizó este resultado considerando sistemas de tareas periódicas, y se halló una cota superior más compleja de calcular, para la utilización de un conjunto de tareas planificadas bajo RM con el esquema particionado, usando la condición de Liu y Layland para verificar que las tareas son planificables en cada procesador. También se demostró que existen algoritmos que, efectivamente, alcanzan esta cota.

Otro resultado importante fue dado en [SLM03]. Aquí, se demuestra que se puede mejorar considerablemente la utilización alcanzable por el esquema particionado bajo RM. Se comparan versiones de RM-FF; una usando la condición de Liu y Layland (LL), y la otra, usando la prueba de factibilidad RBound. Se llega a demostrar que RM-FF con RBound alcanza una utilización planificable en promedio del 96 %, mientras que RM-FF con LL, se alcanza una utilización planificable en promedio del 72 % (para conjuntos de tareas generados aleatoriamente).

En el caso que se suponga que se tiene un número infinito de procesadores, se ha investigado el siguiente problema: ¿cuántos procesadores más necesitaría un algoritmo no óptimo bajo RM usando el esquema particionado en relación al número de procesadores que necesita otro algoritmo óptimo, también bajo RM, para planificar un conjunto de tareas periódicas? Dhall y Liu comenzaron a estudiar este problema en 1978 [DL78], en específico, bajo los algoritmos RM-FF y RM-NFD. Desde ese entonces, muchos otros investigadores han extendido este problema a otros algoritmos de planificación bajo RM utilizando el esquema particionado [ABS95], pero este tema no será desarrollado aquí por razones de espacio.

> Los mejores algoritmos para la planificación multiprocesador de prioridades estáticas bajo el esquema particionado son los RM con alguna heurística *bin-packing* aplicada a las tareas reordenadas en orden decreciente de sus utilizaciones, por ejemplo, RM-FFD y RM-BFD. No se conoce algoritmo óptimo para esta clase [RR07].

#### 2.2.4.2. Esquema particionado con prioridades dinámicas

Al igual que con RM, se pueden definir algoritmos que usen EDF junto con alguna heurística para asignar las tareas a los procesadores[4]. De esta forma, se han definido los algoritmos EDF-FF, EDF-BF, etc. López y otros investigadores hicieron un amplio estudio de los algoritmos de planificación multiprocesador bajo EDF en [JMLG04a]. Ellos proveen una cota superior para la utilización del conjunto de tareas, bajo cualquier algoritmo EDF con esquema particionado (independientemente del esquema de asignación utilizado), y demuestran que EDF-FF y EDF-BF alcanzan esta cota, incluyendo sus variaciones EDF-FFI, EDF-BFI, EDF-FFD y EDF-BFD:

[4]Para la planificación multiprocesador de prioridades dinámicas bajo el esquema particionado, EDF ha sido el algoritmo que más se ha escogido debido a que en un procesador, EDF logra una utilización planificable de 1. Véase la condición necesaria y suficiente de la Ecuación 2.10.

**Teorema 2.2.7.** *[JMLG04a] Sea $\Gamma(n, \alpha)$ la clase de sistemas de $n$ tareas periódicas, en los cuales cada tarea tiene una utilización de a lo sumo $\alpha$. Supóngase que $n > m\beta$ y sean*

$$\beta = \lfloor 1/\alpha \rfloor, \ U_{LL}^{EDF}(n, m, \alpha) = \frac{\beta m + 1}{\beta + 1}. \tag{2.14}$$

*Entonces:*

1. *Existen conjuntos de tareas periódicas $\tau \in \Gamma(n, \alpha)$ con utilización*

$$U(\tau) \geq U_{LL}^{EDF} + \epsilon, \tag{2.15}$$

   *para un $\epsilon > 0$ tan pequeño como se quiera, que no son planificables en $m$ procesadores bajo ninguna planificación basada en EDF, usando el esquema particionado.*

2. *EDF-FF y EDF-BF alcanzan los dos una utilización planificable de $U_{LL}^{EDF}$. Es decir, cada uno puede planificar cualquier sistema de tareas periódicas $\tau \in \Gamma(n, \alpha)$ dado que $U(\tau) \leq U_{LL}^{EDF}$.*

También probaron que EDF usando Worst-Fit no alcanza la cota máxima, sino una utilización planificable menor, de $m - \alpha(m - 1)$; pero si las tareas están ordenadas en orden creciente o decreciente de sus utilizaciones, entonces Worst-Fit sí alcanza la cota máxima de utilización para que el conjunto de tareas sea planificable. Sin embargo, esta cota de utilización en el caso general, cuando $\alpha = 1$ que da el teorema anterior es muy baja, de $(m + 1)/2$ (cercana al 50 %).

Otros autores han investigado el desempeño de la planificación usando EDF bajo un esquema semi-particionado asumiendo que existe un número infinito de procesadores. Andersson y Tovar propusieron el algoritmo EKG[5] [AT06], el cual es un enfoque para planificar conjuntos de tareas periódicas cuyos plazos coinciden con los períodos, basado en el esquema particionado, pero dividiendo en dos algunas instancias de tareas y asignando las partes a diferentes procesadores. La cota de utilización para EKG depende de un parámetro $k$, que se usa para dividir las tareas en dos grupos: uno de tareas pesadas (*heavy*) y otro de tareas ligeras (*light*). La cota de utilización de EKG está dada por:

$$U_{EKG} = \begin{cases} k/(k+1) \text{ si } k < m, \\ 1 \text{ en caso contrario.} \end{cases} \tag{2.16}$$

la cual es de un 66 % para $k = 2$.

| Los mejores algoritmos para la planificación multiprocesador de prioridades dinámicas bajo el esquema particionado son EDF-FF y EDF-BF. Ambos alcanzan una cota de utilización planificable de $(m + 1)/2$, la cual es óptima para esta clase [RR07]. |
|---|

[5] ***E**DF with task splitting and **k** processors in a **g**roup.*

### 2.2.5. Trabajos relacionados

Recientemente, Chattopadhyay y Baruah propusieron un enfoque novedoso basado en tablas precalculadas, para decidir si un conjunto de tareas es planificable en un ambiente homogéneo multiprocesador, basado en un esquema particionado y planificación bajo EDF en cada procesador [CB11]. Según este procedimiento, se resuelve en tiempo polinomial el problema de la planificación, pero además de que no se asegura un particionado óptimo, el cálculo de la tabla consume una gran cantidad de tiempo y cálculos.

A grandes rasgos, se escogen ciertos valores de utilizaciones dependiendo de un valor $\epsilon$ y, exhaustivamente, se prueban todas las posibles combinaciones de distintos valores, determinándose cuáles conforman un particionado factible. Luego, dado un determinado conjunto de tareas, se aproximan las utilizaciones a los valores usados en la tabla y se decide el particionado. Mientras más pequeño sea el $\epsilon$, más precisa será la aproximación, pero también, más trabajoso será el cálculo de la tabla. Este método no es heurístico, pero tampoco es del todo exacto, pues el cálculo de la tabla depende de un parámetro $\epsilon$, y al aproximar el conjunto real de tareas a los valores con los que se calculó la tabla, se pierde precisión.

Un enfoque exacto fue el propuesto en [BB08] donde se modela el problema del particionado como un problema de programación lineal entera binaria. Ellos proponen el modelado de las restricciones para la planificación bajo EDF y FP[6] (*Fixed Priority*) en cada uno de los procesadores y una reducción en el número de restricciones. Originalmente, para representar de forma exacta el problema del particionado, se tiene una cantidad pseudo-polinomial de restricciones. Luego de la reducción, se representa el problema de forma aproximada, con una cantidad polinomial de restricciones.

No obstante, sólo exponen el planteamiento del problema de optimización, pues el objetivo de su trabajo no es desarrollar o aplicar herramientas de optimización para resolver problemas de programación entera binaria. Queda entonces abierto el problema de evaluar la aplicación de técnicas exactas a este tipo de modelos en el campo de la planificación multiprocesador.

### 2.2.6. Esquema global

En el esquema global, se tiene una cola global, y diferentes instancias de una misma tarea pueden ejecutarse en distintos procesadores (véase la Figura 2.2). Al igual que en el esquema particionado, existe un control de admisión que es el que acepta o no las tareas en el sistema. Diferentes instancias de una misma tarea pueden ejecutarse en procesadores distintos. Generalmente, una instancia de una tarea termina su ejecución en el procesador al que fue asignada, pero a veces se permite la migración, es decir, que una instancia de cualquier tarea pueda comenzar la ejecución en un

[6]FP es una de las políticas más populares de planificación en línea [LL73].

procesador y terminar su ejecución en otro al ser expropiada.

Al principio, este esquema fue menos estudiado que el particionado, debido, entre otras razones, a que carece de soporte para modelos más avanzados, como por ejemplo, modelos de manejo de recursos compartidos.

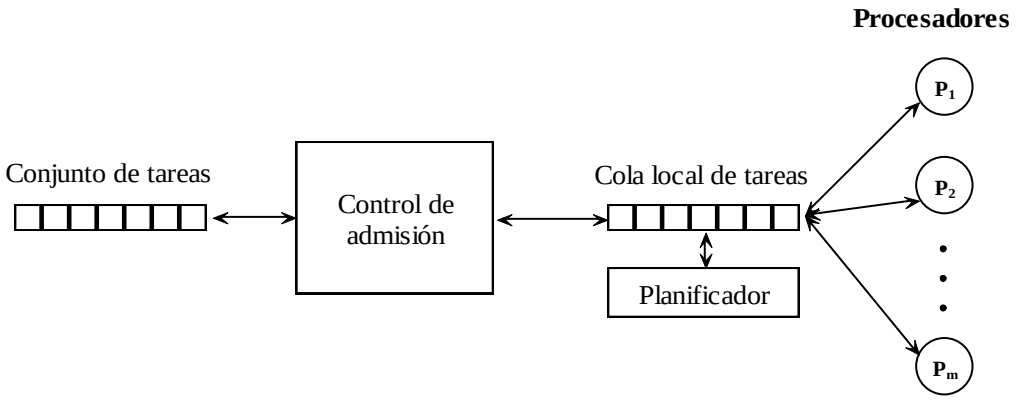


Figura 2.2: Esquema global.

En este esquema, en cada instante de planificación, se selecciona la tarea de mayor prioridad para que se ejecute en alguno de los procesadores, expropiando otra tarea de menor prioridad y usando migración si fuese necesario. Sin perder generalidad, se puede suponer que las tareas están ordenadas de forma decreciente respecto a sus prioridades. De entre todas las tareas que llegan al sistema y que no han completado su ejecución, se escogerán las $n$ tareas con mayor prioridad para ser ejecutadas en los $m$ procesadores. El procesador escogido para cada tarea es arbitrario, y si hay menos de $m$ tareas a ser ejecutadas en un instante de planificación dado, algún procesador estará inactivo.

Algunas de las ventajas del esquema global, en relación al particionado, son las citadas a continuación:

- Típicamente, existe un menor número de cambios de contexto y expropiaciones, porque el planificador sólo expropiará a una tarea cuando no haya procesadores inactivos.

- El hecho de que una tarea a veces no emplea todo el tiempo de ejecución en el peor caso, sino que se ejecuta en menos tiempo, permite que el tiempo sobrante sea utilizado por otras tareas en el sistema.

- Si una tarea sobrepasa su tiempo de ejecución en el peor caso, hay una menor probabilidad de que se incumpla un plazo. Es menos probable que se incurra en

un comportamiento en el peor caso del sistema completo que si esto sucediera en un único procesador.

- No se necesita ejecutar algoritmos de balance de carga y/ó asignación de las tareas cuando cambia el conjunto de tareas.

Debido a esta flexibilidad en cuanto a migración de tareas e instancias de tareas, últimamente se ha tenido la idea de que este tipo de algoritmos pudieran alcanzar cotas de planificación más altas. Por esta razón, en los últimos años, el esquema global en la planificación multiprocesador ha sido objeto de intensa investigación.

#### 2.2.6.1. Esquema global con prioridades estáticas

En [DL78] Dhall y Liu mostraron que la planificación multiprocesador bajo el esquema global presenta una anomalía a la que llamaron el *efecto Dhall*. Esto consiste en que ciertos conjuntos de tareas con utilizaciones muy bajas, son imposibles de planificar ya sea bajo EDF o RM, cuando en realidad son factibles de planificar bajo otros algoritmos. Esto hizo surgir la idea de que el problema con la planificación global radicaba en mezclar las tareas de baja utilización con las tareas de alta utilización, o sea, que las prioridades se asignaran independientemente de las utilizaciones de las tareas. Por esta razón, surgieron algoritmos que agrupan las tareas de acuerdo a sus utilizaciones en pesadas *("heavy")* y ligeras *("light")*.

Andersson y Johnson desarrollaron en [AJ00] un algoritmo de planificación global de prioridades estáticas llamado AdaptiveTKC, el cual evita el *efecto Dhall* por medio de la asignación de una mayor prioridad a las tareas con menor holgura[7] (medida de la diferencia entre el período y el tiempo de ejecución). Ellos demostraron que bajo este algoritmo, se evita el *efecto Dhall*, asumiendo que todas las tareas (hay $m+1$ en total), salvo una, tienen el mismo período y tiempo de ejecución. También formularon una conjetura para la cota superior de utilización planificable del algoritmo. Sin embargo, no ha sido demostrado que este sistema de tareas represente el peor caso, ni que se haya obtenido una cota de utilización para que un conjunto de tareas sea planificable.

Motivados por la misma idea de separar las tareas de acuerdo a su utilización y de dar mayores prioridades a las tareas con utilizaciones más grandes, se propuso en [BAJ03] una variante de RM, un algoritmo de planificación multiprocesador global que asigna mayor prioridad a las tareas con mayor utilización. Dicho algoritmo llamado RM-US$[m/(3m-2)]$ asigna las prioridades de la siguiente forma:

- Si $u_i > \frac{m}{3m-2}$, entonces $\tau_i$ tiene la mayor prioridad[8].
- Si $u_i \leq \frac{m}{3m-2}$, entonces $\tau_i$ tiene prioridad según RM.

[7] A cada tarea periódica $\tau_i$ se le asigna la prioridad $T_i - kC_i$, donde $k \geq 0$ es el factor de holgura global.

[8] Si dos tareas tienen igual prioridad, se escoge una arbitrariamente para ejecución.

Igualmente, se halló una cota de máxima utilización para que un conjunto de tareas periódicas sea planificable bajo RM-US$[m/(3m-2)]$. Pero esta cota es muy baja, de aproximadamente $0.33\bar{3}$ cuando $m \to \infty$, aunque con este algoritmo también se evita el *efecto Dhall*:

**Teorema 2.2.8.** *[BAJ03] Cualquier sistema $\tau$ de tareas periódicas cuya utilización total cumple que $U(\tau) \leq \frac{m^2}{3m-2}$, es planificable en m procesadores bajo el algoritmo RM-US[m/(3m-2)].*

Más tarde, Lundberg mejoró esta cota límite, aumentándola a 0.37482 y demostró que este límite no puede ser mejorado [Lun02]. Otros investigadores [LW82] estudiaron el problema fijando restricciones como por ejemplo, que las utilizaciones de todas las tareas no superen un cierto valor, o que los períodos de las tareas no coincidan con sus plazos. Baker, por ejemplo, desarrolló dos pruebas suficientes de planificación para la planificación global de conjuntos de tareas cuyos plazos difieren de sus períodos bajo el algoritmo DM[9] [Bak03a][Bak03b]. Un corolario de una de esas pruebas es el siguiente:

**Corolario 2.2.1.** *[Bak03a][Bak03b] Un conjunto $\tau$ de tareas periódicas cuyos plazos coinciden con los períodos es planificable en m procesadores bajo el algoritmo global RM[10] si:*

$$U(\tau) \leq \frac{m}{2}(1-\alpha) + \alpha, \tag{2.17}$$

*donde* $\alpha = \text{máx}\{u_i \mid 1 \leq i \leq n\}$.

En particular, cuando todas las tareas tienen una utilización de a lo sumo 1/3, este corolario da una cota de utilización planificable de $\frac{m}{3} + \frac{1}{3}$, la cual es ligeramente mejor que la cota dada por el siguiente teorema:

**Teorema 2.2.9.** *[BG03]Cualquier conjunto de tareas periódicas $\tau$, donde se cumpla que $u_i \leq \frac{1}{3}\ \forall i$, es planificable en m procesadores bajo el algoritmo global RM si se cumple que $U(\tau) \leq \frac{m}{3}$.*

El mejor algoritmo en la planificación multiprocesador de prioridades estáticas bajo el esquema global es RM-US$[m/(3m-2)]$, el cual alcanza una utilización planificable de $m^2/(3m-2)$. No se conoce aún ningún algoritmo óptimo para esta clase [RR07].

[9] *Deadline-Monotonic* es un algoritmo óptimo de planificación uniprocesador de prioridades estáticas que asigna las prioridades de forma tal que la tarea con menor plazo relativo (a su activación) tiene mayor prioridad [LW82]. Se aplica solamente a los conjuntos de tareas cuyos plazos difieren de los períodos.

[10] Cuando los plazos coinciden con los períodos, RM coincide con DM.

#### 2.2.6.2. Esquema global con prioridades dinámicas

En [SFB03], Funk, Baruah y Goosens hallaron una cota superior para la utilización planificable de un conjunto de tareas bajo EDF usando el esquema global en múltiples procesadores:

**Teorema 2.2.10.** *[SFB03] Cualquier conjunto $\tau$ de tareas periódicas es planificable en $m$ procesadores bajo el algoritmo global EDF si se cumple que $U(\tau) \leq m - \alpha(m-1)$, donde $\alpha = \max\{u_i \mid 1 \leq i \leq n\}$.*

Esta cota disminuye en la misma medida que aumenta la máxima utilización de las tareas. Esto indica que, al igual que con RM, el efecto *Dhall* puede presentarse en la planificación global bajo EDF. Baker generalizó este resultado al caso de conjuntos de tareas periódicas cuyos plazos difieren de los períodos, y presentó dos pruebas suficientes de planificación para la planificación global en múltiples procesadores bajo EDF [Bak03b] .

Para evitar el *efecto Dhall*, se modificó la asignación de prioridades dinámicas de manera similar a la variante que se propuso en [BAJ03], mencionada anteriormente (RM-US$[m/(3m-2)]$). En [SB02] se adapta este algoritmo y se presenta un algoritmo de planificación global de prioridades dinámicas al que se le llamó EDF-US$[m/(2m-1)]$. Este algoritmo asigna las prioridades análogamente a como lo hace el algoritmo RM-US$[m/(3m-2)]$, sólo que en vez de asignárseles las prioridades según RM a las tareas cuya utilización sea menor o igual que $m/(3m-2)$, se les asigna según EDF. En este trabajo, se halla una cota superior para la utilización del conjunto de tareas, cuyo límite se acerca a $(m+1)/2$ cuando el número de procesadores tiende a infinito.

Otro de los resultados importantes lo obtuvo Baruah en 2004, cuando desarrolló el algoritmo llamado fpEDF, que primero ordena las tareas en orden no creciente de sus utilizaciones y luego a ciertas tareas les da mayor prioridad y a las otras, les asigna las prioridades correspondientes a EDF. Baruah demostró que fpEDF es óptimo pues su cota máxima de utilización planificable coincide con $(m+1)/2$, que es la cota superior máxima de utilización planificable para cualquier algoritmo de planificación de prioridades dinámicas a nivel de instancias de tareas [Bar04].

| El mejor algoritmo en la planificación multiprocesador de prioridades dinámicas bajo el esquema global es fpEDF, el cual alcanza una utilización planificable de $(m+1)/2$ (óptima para esta clase) [RR07]. |
|---|

## 2.2.7. Planificación con prioridades plenamente dinámicas

El algoritmo Pfair (*Proportionate Fairness*) fue introducido por Baruah en 1996 [SBV96]. Se trata de un algoritmo de generación de planes de ejecución, el cual es aplicable a conjuntos de tareas periódicas cuyos plazos coinciden con los períodos. Este algoritmo está basado en la idea de una planificación fluida, donde cada tarea progresa proporcionalmente a su utilización. A cada tarea se le asigna un peso igual

a su utilización, el cual define la frecuencia con la cual va a ser planificada. La planificación Pfair divide la línea de tiempo en cuantos de igual longitud. En cada cuanto de tiempo $t$, el planificador asigna tareas a los procesadores, de tal forma que el tiempo de procesador acumulado asignado a cada tarea $\tau_i$ es o bien $\lceil tu_i \rceil$ ó $\lfloor tu_i \rfloor$ en el intervalo $[0, t)$.

Aunque el algoritmo Pfair es óptimo para el tipo de conjuntos de tareas mencionado anteriormente, pues tiene una cota de máxima utilización igual a $m$ [SBV96], tomando decisiones de planificación en cada cuanto de tiempo, se incurre en una alta sobrecarga operativa, en la práctica.

Los algoritmos de planificación PFair no son necesariamente conservadores del trabajo[11] Existen numerosas variantes de este algoritmo y otros algoritmos similares (para más información, véase [DB09][RR07]-Capítulo 24) y se resumen algunas características en la siguiente Tabla:

| Algoritmo | Autores | Características |
|---|---|---|
| Pfair | Baruah et al. [SBV96] | Puede que haya un procesador disponible para ejecución, y no se aproveche. |
| ERFair | Anderson, Srinivasan [AS00a] | (*Early-Release fair*) Variante de PFair. A diferencia de Pfair, ERFair es un algoritmo conservador del trabajo. |
| PD | Baruah et al. [SKBP95] | (*Pseudo-deadline*) Separa las tareas en pesadas ($u_i > 0.5$) y ligeras, mejorando así la eficiencia del enfoque Pfair. |
| $PD^2$ | Anderson, Srinivasan [AS01] | PD mejorado. Actualmente es el más eficiente de los algoritmos de planificación Pfair. |
| EPDF | Anderson, Srinivasan [AS00b] | (*Earliest pseudo-deadline first*) Es una variante de PD aplicada a tareas esporádicas cuyos plazos coincidan con los períodos, es óptimo para 2 procesadores, pero no para más. |
| BF | Zhu et al. [DZM03] | (*Boundary Fair*) Similar a Pfair. Sólo hace decisiones de planificación en las fronteras de los períodos. Es óptimo para tareas periódicas cuyos plazos coinciden con los períodos, y requiere 25-50 % de los instantes de planificación que PD requiere. |

Tabla 2.1: El algoritmo Pfair y algunas de sus variantes.

[11] (*Work-conserving algorithm*) Un algoritmo de planificación conservador del trabajo es aquel que nunca deja inactivo un procesador cuando hay instancias de tareas en el sistema que pudieran ejecutarse en ese procesador.

# Capítulo 3

# Programación Lineal

Uno de los adelantos más importantes en las matemáticas aplicadas en el siglo XX, desde el punto de vista de la economía, es la programación lineal. Muchos problemas prácticos de la investigación de operaciones pueden plantearse como problemas de programación lineal. Ejemplos de estos problemas son aquellos que surgen en los procesos industriales, la agricultura, la transportación aérea y terrestre, las finanzas, la producción y exploración de petróleo, las comunicaciones, el procesamiento de alimentos, la distribución de recursos naturales, etc.

Técnicamente hablando, la programación lineal es una técnica matemática de optimización, que maximiza (ó minimiza) una función objetivo, sujeta a una serie de restricciones impuestas por la naturaleza del problema que se esté resolviendo. En general, estos problemas se plantean de la forma estándar:

$$\min_{x_1,\ldots,x_n} f(X) = C^T X \tag{3.1}$$

$$s.a.\ AX \geq B \tag{3.1a}$$

$$x_i \geq 0 \quad \forall i \in [1, n], \tag{3.1b}$$

donde $C$ y $X$ son vectores en $\mathbb{R}^n$, $B$ es un vector de $\mathbb{R}^m$, y $A$ es una matriz de $m \times n$.

El matemático francés Jean Baptiste-Joseph Fourier (1768-1830) fue el primero en intuir de forma imprecisa los métodos de lo que actualmente se conoce como programación lineal. Mucho más tarde, en 1939, el matemático ruso Leonidas Vitalyevich Kantorovich[1] publicó una extensa monografía titulada *Métodos matemáticos de organización y planificación de la producción*, en la que por vez primera se hace corresponder a una extensa gama de problemas una teoría precisa y bien definida que es hoy en día la programación lineal.

En los años posteriores a la Segunda Guerra Mundial, en Estados Unidos se asumió que la eficaz coordinación de los recursos de la nación era un problema de tal complejidad, que su resolución necesitaba forzosamente de los modelos de optimización

[1]Premio Nobel de Economía en 1975.

que resuelve la programación lineal. Fue en 1947 que George Dantzig publicó el algoritmo simplex y formuló, en términos matemáticos muy precisos, el enunciado estándar al que cabe reducir todo problema de programación lineal. Se le considera el fundador de la técnica junto a John Von Neumann, quien desarrolló la teoría de la dualidad en el mismo año.

## 3.1. Terminología

- Al problema estándar de programación lineal modelado por las ecuaciones 3.1, 3.1a-b se le denota comúnmente como (LP) (*Linear Program*).
- La función $f(X)$ que debe ser maximizada o minimizada se llama función objetivo (*objective function*). Problemas ejemplo: minimizar el costo de la producción, ó el costo de la transportación, maximizar la ganancia, etc.
- Para un problema estándar de programación lineal (LP)[2], se dice que el punto $\bar{X}$ es factible (*feasible*) si satisface las restricciones del problema (3.1a, 3.1b).
- Se dice que un problema de programación lineal (LP) es factible (*feasible*) si el conjunto de puntos factibles es no vacío. De otra forma, se dice que el problema no tiene solución o es no factible (*infeasible*).
- Se dice que un (LP) de maximización (minimización) es no acotado si la función objetivo puede tomar valores positivos (negativos) arbitrariamente grandes, evaluada en vectores factibles. De otra forma, se dice que el problema es acotado. Geométricamente, la región factible de un problema (LP) acotado está definida por un polítopo $n$-dimensional[3].
- El máximo (mínimo) de un (LP) es el mayor (menor) valor que la función objetivo puede tomar, tomando este máximo (mínimo) sobre todos los puntos factibles del problema.
- Un punto factible en el cual la función objetivo alcanza su valor máximo (mínimo) se denomina óptimo (*optimal value*).
- Un problema de programación lineal puede tener:

    1. Una única solución óptima que corresponde a un único punto de la región factible, sin importar si ésta es o no acotada.
    2. Infinitas soluciones óptimas: existe un conjunto infinito de puntos óptimos (la función objetivo evaluada en todos ellos arroja el mismo valor máximo (mínimo)). En dos dimensiones, estos puntos se encuentran a lo largo de un segmento de recta. Esto puede suceder independientemente de si la región factible es acotada o no.

[2]La forma estándar cambia, según la literatura.

[3]$n$: número de variables

3. Ninguna solución óptima: la región factible es vacía.
4. No existe solución óptima acotada: cuando una ó más variables del problema pueden aumentar o disminuir infinitamente su valor, mejorando constantemente el valor de la función objetivo.

## 3.2. Problema dual

Para cada problema de programación lineal (LP), existe otro problema de optimización estrechamente relacionado con él, al que se le llama **problema dual**. El dual del problema representado por las ecuaciones 3.1, 3.1a-c es el siguiente:

$$\max_{x_1,\ldots,x_n} Y^T B \tag{3.2}$$

$$s.a.\ Y^T A \geq C \tag{3.3a}$$

$$y_j \geq 0 \quad \forall j \in [1, m]. \tag{3.3b}$$

Si un (LP) es de minimización, el problema dual será de maximización, y viceversa. El vector $B$ de términos independientes de las restricciones del problema original se convierte en el vector de la función objetivo en el problema dual, y viceversa. En el problema dual hay tantas variables como restricciones hay en el problema original (también llamado primal), y viceversa. El dual del problema de la Wyndor Glass Co. (véase la página siguiente) es el siguiente:

$$\begin{aligned} &\min 18y_1 + 4y_2 + 12y_3 \\ &s.a.\ 3y_1 + y_2 \geq 3 \\ &\quad 2y_1 + 2y_3 \geq 5 \\ &\quad y_1 \geq 0,\ y_2 \geq 0,\ y_3 \geq 0. \end{aligned} \tag{3.3}$$

Los resultados fundamentales que relacionan un (LP) y su dual son los que se establecen a continuación:

**Teorema 3.2.1.** *[Rao96] El dual del dual coincide con el problema original (primal).*

**Teorema 3.2.2.** *[Rao96] (Dualidad débil) Sean $\hat{X}$ y $\hat{Y}$ soluciones factibles para los problemas primal (de maximización) y dual (de minimización), respectivamente. Entonces se cumple que $C^T\hat{X} \leq \hat{Y}^T B$.*

**Corolario 3.2.1.** *Si las soluciones factibles $\hat{X}$ y $\hat{Y}$ del primal (de maximización) y del dual (de minimización) cumplen que $C^T\hat{X} = \hat{Y}^T B$, entonces $\hat{X}$ y $\hat{Y}$ son soluciones óptimas del primal y del dual, respectivamente.*

**Corolario 3.2.2.** *Si los problemas primal (de maximización) y dual (de minimización) son factibles ambos, entonces ambos tienen solución óptima acotada.*

**Teorema 3.2.3.** *[Rao96] (Dualidad Fuerte) Si el problema primal (dual) tiene una solución óptima acotada, entonces el dual (primal) también la tiene, y se cumple que:*

$$\max\ C^T X = \min\ Y^T B. \tag{3.4}$$

## 3.3. El método Simplex

Considérese el problema clásico de la Wyndor Glass Co.[HL05][4]:

$$\begin{aligned} &\text{máx } 3x_1 + 5x_2 \\ &s.a.\ 3x_1 + 2x_2 \leq 18 \\ &\qquad x_1 \leq 4 \\ &\qquad 2x_2 \leq 12 \\ &x_1 \geq 0,\ x_2 \geq 0 \end{aligned} \tag{3.5}$$

En la Figura 3.1 el área sombreada corresponde a la región convexa del conjunto de todos los puntos factibles para este problema; es decir, el conjunto de puntos que satisfacen las restricciones del problema. Dentro de este conjunto de soluciones, se pretende encontrar aquella que maximice el valor de la función objetivo.

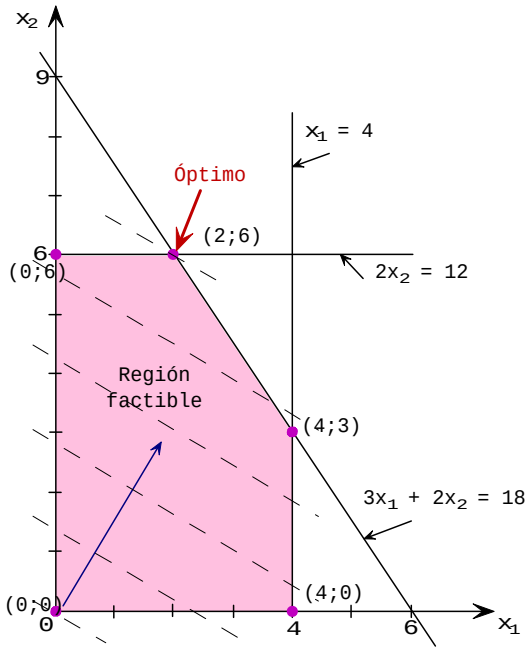


Figura 3.1: Región factible para el problema de la Wyndor Glass Co.

Todos aquellos puntos que generan un mismo valor de la función objetivo deben satisfacer la ecuación $C^T X = z$. Si se desea maximizar dicho valor, debemos mover paralelamente a sí mismo éste plano (que en dos dimensiones corresponde a una recta) en la dirección que permita disminuir el valor de la función objetivo hasta donde sea posible, de manera que no se abandone el conjunto de soluciones factibles.

[4]Una empresa que produce artículos de vidrio de alta calidad. Es un ejemplo prototipo de programación lineal, estudiado en Investigación de Operaciones.

Esta dirección está dada por el vector $(3; 5)$ que corresponde a la función objetivo (véase la Figura 3.1).

El algoritmo simplex comienza en un vértice y genera una sucesión de iteraciones factibles, moviéndose desde un vértice del conjunto factible al vértice adyacente con menor valor de la función objetivo, hasta que alcanza el vértice de la solución óptima, permitiendo además identificar cuando no existe un óptimo finito o la región factible es vacía [AHL60]. Para el ejemplo 3.5, el método simplex parte del vértice $(0; 0)$, luego se mueve al vértice $(0; 6)$ y por último, alcanza el vértice $(2; 6)$ que es el óptimo global en este ejemplo (Figura 3.2).

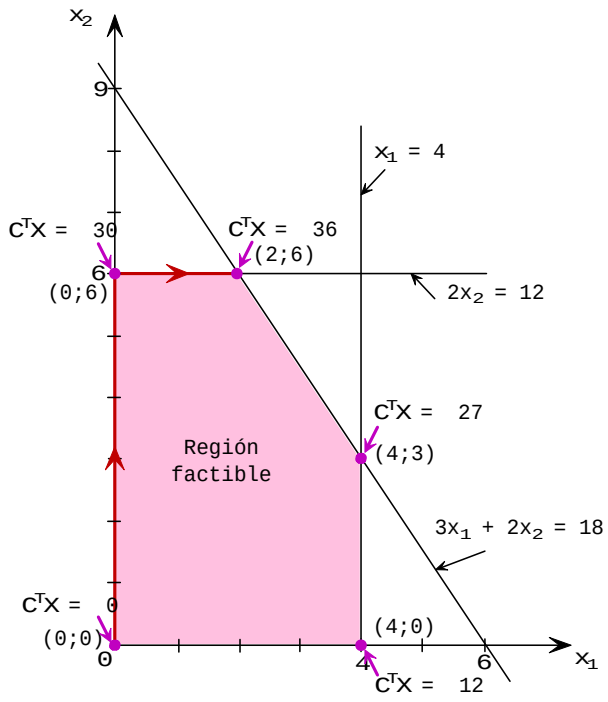


Figura 3.2: Iteraciones del método Simplex para el problema de la Wyndor Glass Co.

Algebraicamente, el fundamento del método es mejorar iterativamente el valor de la función objetivo, revisando sistemas de ecuaciones (organizadas en una tabla) equivalentes al sistema original de desigualdades que describe la región factible. No se explicará aquí el procedimiento iterativo del método simplex, pero se puede encontrar en numerosas bibliografías como [NW06, HL05, Rao96], por citar algunas.

También existen varias versiones de este método, como por ejemplo el Simplex Dual (*Dual Simplex*[5]) y el Simplex Revisado (*Revised Simplex*[6]). El método Simplex parte

[5]Formulado por Lemke en 1954.
[6]Formulado por Dantzig en 1953.

de una solución factible pero no óptima, e iterativamente genera soluciones factibles cada vez mejores hasta encontrar la solución óptima, si existe. Es decir, la lógica del método es mantener la factibilidad de las soluciones. El Simplex Dual se aplica al problema dual, pero se desarrolla de forma tal que se pueda usar la misma tabla que se utiliza en el Simplex primal. Se parte de una solución óptima, pero no factible, e iterativamente se busca la factibilidad. Con este método también se alcanza la solución óptima, si ésta existe. Además, en muchos casos, puede obtenerse la solución óptima de un (LP) de manera mucho más simple, resolviendo el problema dual asociado [Rao96].

El método simplex fue una de las primeras herramientas que surgió para resolver problemas de optimización lineal entera, y ha sido ampliamente usado. Sin embargo, existen problemas donde este algoritmo tiene un desempeño muy pobre, llegando a pasar por todos los vértices del polítopo ($2^n$, donde $n$ es el número de variables) antes de alcanzar la solución óptima [NW06]. Esto demostró que la complejidad del método simplex puede ser exponencial en la dimensión del problema y la comunidad de optimización se dedicó a buscar otros algoritmos de programación lineal que tuviesen complejidad polinomial.

## 3.4. Métodos de Punto Interior

En los años 80, Narendra Karmarkar presentó un algoritmo polinomial que se aproxima a la solución óptima a través del interior del polítopo en vez de moverse por los vértices de su frontera como lo hace el método Simplex [Kar84] (véase la Figura 3.3 que ejemplifica la diferencia entre el método Simplex y uno de punto interior, para un caso tridimensional). Esto marcó el inicio de la investigación en este tipo de métodos, a los que se les llamó **de punto interior** (*interior-point methods*) [Kar84].

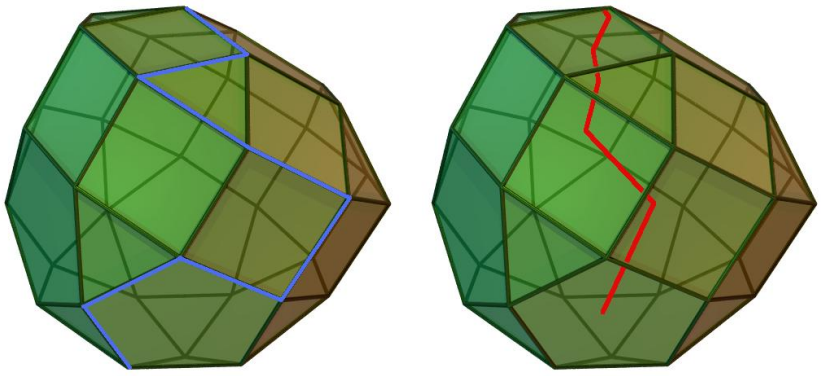

Figura 3.3: Método simplex Vs. método de puntos interiores (polítopo tomado de *www.wikipedia.com*).

La estrategia de un algoritmo de punto interior es la siguiente: una vez conocida la región de factibilidad y un punto inicial e interior a esta región, se debe definir un modo de progreso en una dirección que permita mejorar el valor de la función objetivo, de manera tal que el nuevo punto siga siendo factible e interior. Este proceso se debe repetir hasta que no pueda haber una mayor mejora en el valor de la función objetivo. El término "interior"significa que este método se acerca a la frontera del conjunto factible solamente en el límite. Se acercan a la solución desde el interior o el exterior de la región factible, pero nunca tocan su frontera.

Los algoritmos de punto interior se dividen en los siguientes tipos principales, aunque no se dedicará espacio aquí a explicar cada uno de ellos:

Métodos de Punto Interior
- ⋆Path-Following
  - •Short-Step
  - •Long-Step
  - •Predictor-Corrector
- ⋆Affine Scaling
- ⋆Potential Reduction
- ⋆Cutting Plane

### 3.4.1. Algoritmo Predictor-Corrector de Mehrotra

En esta sección se describe un algoritmo de punto interior de tipo predictor-corrector, que en la actualidad es la base de muchas implementaciones de software. Es un algoritmo primal-dual, lo que significa que se calculan las soluciones del problema primal y dual, al mismo tiempo.

Considérese ahora un (LP) y su dual en la siguiente forma estándar:

$$\begin{array}{ll} \text{mín } C^T X, & \text{máx } B^T Y, \\ s.a.\ AX = B, & s.a.\ A^T Y + S = C, \\ X \geq 0. & Y \geq 0. \end{array} \tag{3.6}$$

Las soluciones primal-duales de 3.6 cumplen el siguiente sistema de condiciones de Karush-Kuhn-Tucker (KKT) con $F : \mathbb{R}^{2n+m} \to \mathbb{R}^{2n+m}$ [NW06]:

$$F(X, Y, S) = \begin{bmatrix} A^T Y + S - C \\ AX - B \\ X^d S^d e \end{bmatrix} = 0, \tag{3.7}$$

$$X, S \geq 0, \tag{3.8}$$

donde $X^d = diagonal(x_1, \ldots, x_n)$, $S^d = diagonal(s_1, \ldots, s_n)$ y $e = (1, 1, \ldots, 1)^T$. Los métodos primal-duales generan puntos $(X^k, Y^k, S^k)$ en cada iteración, que satisfacen la ecuación 3.8 en el sentido estricto. Las soluciones de 3.6 se obtienen resolviendo el sistema anterior mediante variantes del método de Newton, de manera que $(X, S) > 0$. Como el sistema 3.7 es no lineal, se aplica el método de Newton para sistemas no

lineales, con lo cual la dirección de búsqueda $(\Delta X, \Delta Y, \Delta S)$ se halla resolviendo el siguiente sistema, donde $J$ es el jacobiano de $F$ y el punto $(X, Y, S)$ cumple que $(X, S) > 0$:

$$\begin{bmatrix} 0 & A^T & I \\ A & 0 & 0 \\ S^d & 0 & X^d \end{bmatrix} \begin{bmatrix} \Delta X \\ \Delta Y \\ \Delta S \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ -X^d S^d e \end{bmatrix}. \tag{3.9}$$

El paso en esta dirección no se puede hacer completo, porque violaría la restricción $(X, S) > 0$, así que el nuevo punto es $(X, Y, S) + \alpha(\Delta X, \Delta Y, \Delta S)$, y $\alpha$ se halla haciendo una búsqueda lineal en la dirección de Newton. Sin embargo, la dirección de Newton pura no ofrece mucho progreso en la búsqueda de la solución óptima. Es por esto que los métodos primal-duales sesgan la dirección de búsqueda dentro del ortante $(X, S) > 0$ de manera tal que se pueda avanzar más en la dirección, y al mismo tiempo, no dejan que los componentes de $(X, S)$ se acerquen mucho a la frontera del ortante.

El camino central $\mathcal{C}$ es un arco de puntos estrictamente positivos que está parametrizado por un escalar $\tau > 0$. Cada punto $(X_\tau, Y_\tau, S_\tau)$ en $\mathcal{C}$ cumple estrictamente la desigualdad 3.8 y además, el sistema 3.7 con la diferencia que $X^d S^d e = \tau e$. De esta forma, también se puede definir $\mathcal{C}$ como:

$$F(X_\tau, Y_\tau, S_\tau) = \begin{bmatrix} 0 \\ 0 \\ \tau e \end{bmatrix}, \ (X_\tau, S_\tau) > 0. \tag{3.10}$$

Si $\mathcal{C}$ converge a algún punto cuando $\tau$ tiende a cero, entonces debe converger a una solución primal-dual de 3.6. Lo que hacen los algoritmos primal-duales es tomar pasos de Newton hacia puntos en $\mathcal{C}$ para los cuales $\tau > 0$ en vez de tomar la dirección pura de Newton. Esto se explica porque los pasos de Newton hacia los puntos en $\mathcal{C}$ son sesgados hacia el interior del ortante definido por $(X, S) > 0$, y por consiguiente se pueden tomar pasos más grandes en esta dirección antes de violar las condiciones de positividad.

Esta dirección de búsqueda sesgada se describe de la siguiente forma: se introduce un parámetro de centrado $\sigma \in [0; 1]$ y una medida de dualidad $\mu$:

$$\mu = \frac{1}{n} \sum_{i=1}^{n} x_i s_i = \frac{X^T S}{n}. \tag{3.11}$$

Poniendo $\tau = \sigma\mu$ y aplicando nuevamente el método de Newton al sistema 3.10, el paso $(\Delta X, \Delta Y, \Delta S)$ es un paso de Newton hacia el punto $(X_{\sigma\mu}, Y_{\sigma\mu}, S_{\sigma\mu}) \in \mathcal{C}$ (y no hacia el punto que satisface las condiciones de KKT), y se halla resolviendo:

$$\begin{bmatrix} 0 & A^T & I \\ A & 0 & 0 \\ S^d & 0 & X^d \end{bmatrix} \begin{bmatrix} \Delta X \\ \Delta Y \\ \Delta S \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ -X^d S^d e + \sigma\mu e \end{bmatrix}. \tag{3.12}$$

Hasta ahora se ha asumido que el punto inicial $(X^0, Y^0, S^0)$ es estrictamente factible, pero la mayoría de las veces un punto así es difícil de encontrar. Si se definen los residuos para las dos ecuaciones lineales del sistema de KKT como:

$$r_B = AX - B, \quad r_C = A^T Y + S - C, \tag{3.13}$$

entonces, el sistema de ecuaciones 3.12 para hallar el paso se convierte en:

$$\begin{bmatrix} 0 & A^T & I \\ A & 0 & 0 \\ S^d & 0 & X^d \end{bmatrix} \begin{bmatrix} \Delta X \\ \Delta Y \\ \Delta S \end{bmatrix} = \begin{bmatrix} -r_C \\ -r_B \\ -X^d S^d e + \sigma \mu e \end{bmatrix}. \tag{3.14}$$

Las características principales del algoritmo predictor-corrector de Mehrotra [Meh92] son dos: (1) se añade un corrector a la dirección de búsqueda para que el algoritmo siga una trayectoria más cercana al conjunto de soluciones de 3.6 y (2) el parámetro de centrado se escoge luego de calcular la dirección de búsqueda. En cada iteración, se calcula el predictor (que es la dirección de búsqueda pura de Newton) y se evalúa cuán útil resulta como dirección de búsqueda. Si esta dirección provoca que $\mu$ se reduzca sin violar las condiciones de positividad $(X, S) > 0$, se necesita poco centrado y por consiguiente, se toma $\sigma$ cercano a cero, para luego calcular una dirección de búsqueda centrada con este valor de $\sigma$. Si, por el contrario, la dirección pura de Newton no es productiva, entonces se escoge el parámetro de centrado $\sigma$ cercano a 1.

Primero se calcula el predictor $(\Delta X^{aff}, \Delta Y^{aff}, \Delta S^{aff})$ poniendo $\sigma = 0$ en la ecuación 3.14:

$$\begin{bmatrix} 0 & A^T & I \\ A & 0 & 0 \\ S^d & 0 & X^d \end{bmatrix} \begin{bmatrix} \Delta X^{aff} \\ \Delta Y^{aff} \\ \Delta S^{aff} \end{bmatrix} = \begin{bmatrix} -r_C \\ -r_B \\ -X^d S^d e \end{bmatrix}. \tag{3.15}$$

Como las matrices $A, X^d$ y $S^d$ son grandes y esparcidas, este sistema de ecuaciones se reformula para poder trabajar con matrices más pequeñas, compactas y simétricas:

$$\begin{aligned} S^d \Delta X^{aff} + X^d \Delta S^{aff} &= -X^d S^d e, \\ X^d \Delta S^{aff} &= -S^d \Delta X^{aff} - X^d S^d e, \\ (X^d)^{-1} X^d \Delta S^{aff} &= -(X^d)^{-1} S^d \Delta X^{aff} - (X^d)^{-1} X^d S^d e, \\ \Delta S^{aff} &= -(X^d)^{-1} S^d \Delta X^{aff} - S^d e, \\ \Delta S^{aff} &= -S - (X^d)^{-1} S^d \Delta X^{aff}. \end{aligned} \tag{3.16}$$

De la segunda y primera filas de la matriz del sistema 3.15 se obtienen las siguientes ecuaciones:

$$A \Delta X^{aff} = -r_B, \tag{3.17}$$

$$A^T \Delta Y^{aff} + \Delta S^{aff} = -r_C. \tag{3.18}$$

Sustituyendo 3.16 en 3.18:

$$\begin{aligned} A^T\Delta Y^{aff} - (S + (X^d)^{-1}S^d\Delta X^{aff}) &= -r_C, \\ A^T\Delta Y^{aff} - (X^d)^{-1}S^d\Delta X^{aff} &= -r_C + S. \end{aligned} \tag{3.19}$$

Teniendo en cuenta que las matrices $S^d$ y $X^d$ son diagonales y no singulares, denotemos:

$$D = (S^d)^{-\frac{1}{2}}(X^d)^{\frac{1}{2}}. \tag{3.20}$$

El producto de matrices diagonales es conmutativo:

$$\begin{aligned} D^2 &= (S^d)^{-1}X^d, \\ D^{-2} &= ((S^d)^{-1}X^d)^{-1} = (X^d)^{-1}S^d. \end{aligned} \tag{3.21}$$

Entonces, se tiene la ecuación:

$$A^T\Delta Y^{aff} - D^{-2}\Delta X^{aff} = -r_C + S. \tag{3.22}$$

De las ecuaciones 3.16, 3.17 y 3.22 se obtiene el siguiente sistema equivalente a 3.15:

$$\begin{bmatrix} 0 & A \\ A^T & -D^{-2} \end{bmatrix} \begin{bmatrix} \Delta Y^{aff} \\ \Delta X^{aff} \end{bmatrix} = \begin{bmatrix} -r_B \\ -r_C + S \end{bmatrix}, \tag{3.23}$$

$$\Delta S^{aff} = -S - (X^d)^{-1}S^d\Delta X^{aff}. \tag{3.24}$$

Para evitar resolver también este sistema de ecuaciones, se puede eliminar $\Delta X^{aff}$ del mismo. Para esto, se multiplica la ecuación 3.22 por $AD^2 = A(S^d)^{-1}X^d$ por la izquierda:

$$\begin{aligned} AD^2A^T\Delta Y^{aff} - A(S^d)^{-1}X^d(X^d)^{-1}S^d\Delta X^{aff} &= -AD^2r_C + A(S^d)^{-1}X^dS, \\ AD^2A^T\Delta Y^{aff} - A\Delta X^{aff} &= A(-(S^d)^{-1}X^dr_C + (S^d)^{-1}X^dS^de). \end{aligned}$$

Sustituyendo $A\Delta X^{aff} = -r_B$, se obtiene:

$$\begin{aligned} AD^2A^T\Delta Y^{aff} &= -r_B + A(-(S^d)^{-1}X^dr_C + (S^d)^{-1}S^dX^de), \\ AD^2A^T\Delta Y^{aff} &= -r_B + A(-(S^d)^{-1}X^dr_C + X). \end{aligned} \tag{3.25}$$

De la ecuación 3.18 se puede despejar $\Delta S^{aff}$:

$$\Delta S^{aff} = -r_C - A^T\Delta Y^{aff}. \tag{3.26}$$

Finalmente, de la ecuación 3.16 se tiene:

$$\begin{aligned} (X^d)^{-1}S^d\Delta X^{aff} &= -S - \Delta S^{aff}, \\ (S^d)^{-1}X^d(X^d)^{-1}S\Delta X^{aff} &= -(S^d)^{-1}X^dS - (S^d)^{-1}X^d\Delta S^{aff}, \\ \Delta X^{aff} &= -(S^d)^{-1}X^dS^de - (S^d)^{-1}X^d\Delta S^{aff}, \\ \Delta X^{aff} &= -X - (S^d)^{-1}X^d\Delta S^{aff}. \end{aligned} \tag{3.27}$$

Las ecuaciones 3.25, 3.26 y 3.27 forman el siguiente sistema equivalente a 3.15:

$$\begin{aligned} AD^2A^T\Delta Y^{aff} &= -r_B + A(-(S^d)^{-1}X^d r_C + X), \\ \Delta S^{aff} &= -r_C - A^T\Delta Y^{aff}, \\ \Delta X^{aff} &= -X - (S^d)^{-1}X^d\Delta S^{aff}. \end{aligned} \tag{3.28}$$

La mayoría de las implementaciones de cualquier método primal-dual se basa en sistemas de este tipo, que usan algoritmos de Cholesky para factorizar matrices esparcidas en un producto $LU$. Con ayuda de esta factorización, se obtiene luego fácilmente la incógnita $\Delta Y$ gracias a que las matrices L y U son triangulares. Debe tenerse cuidado, no obstante, al aplicar la factorización de Cholesky porque la matriz $AD^2A^T$ puede ser singular o cercana a singular (*ill-conditioned*) [NW06].

En este punto, ya se ha calculado el predictor $(\Delta X^{aff}, \Delta Y^{aff}, \Delta S^{aff})$. Luego, hay que calcular cuánto se puede ir en esta dirección sin violar las condiciones de no negatividad que deben cumplir $X$ y $S$, es decir $(X, S) \geq 0$ y con una cota superior de 1. Esto se denotará por $\alpha_{aff}^{pri}$ y $\alpha_{aff}^{dual}$. Los superíndices *pri* y *dual* hacen referencia a que se tomarán diferentes longitudes de paso para las variables del problema primal y dual[7]:

$$\alpha_{aff}^{pri} = \text{mín}\left(1, \min_{i:\Delta x_i^{aff}<0} -\frac{x_i}{\Delta x_i^{aff}}\right), \tag{3.29}$$

$$\alpha_{aff}^{dual} = \text{mín}\left(1, \min_{i:\Delta s_i^{aff}<0} -\frac{s_i}{\Delta s_i^{aff}}\right). \tag{3.30}$$

Se define $\mu_{aff}$ como el valor de $\mu$ que se obtendría con un paso completo hacia la frontera:

$$\mu_{aff} = \frac{(X + \alpha_{aff}^{pri}\Delta X^{aff})^T(S + \alpha_{aff}^{dual}\Delta S^{aff})}{n}. \tag{3.31}$$

Se le asigna al parámetro de centrado $\sigma$ el valor:

$$\sigma = \left(\frac{\mu_{aff}}{\mu}\right)^3. \tag{3.32}$$

Este valor tiene la siguiente propiedad: cuando se hace un buen progreso en la dirección del predictor, $\mu_{aff} \ll \mu$, y por ende, $\sigma$ es un valor pequeño (se necesita poco centrado) y viceversa.

El corrector se obtiene reemplazando el lado derecho de la ecuación 3.15 por el vector $(0, 0, -(\Delta X^d)^{aff}(\Delta S^d)^{aff}e)$. Por otra parte, el centrado requiere el vector $(0, 0, \sigma\mu e)$. El paso completo de Mehrotra incluye los componentes del predictor, el corrector y el centrado, y lo que se hace es añadir los tres vectores correspondientes para obtener

[7]Es equivalente plantear: $\alpha_{aff}^{pri} = \text{máx}\{\alpha \in [0;1] \mid x_i + \alpha\Delta x_i^{aff} \geq 0\ \forall i \in [1;n]\}$, y la fórmula análoga para $\alpha_{aff}^{dual}$.

de esta forma el siguiente sistema de ecuaciones[8]:

$$\begin{bmatrix} 0 & A^T & I \\ A & 0 & 0 \\ S^d & 0 & X^d \end{bmatrix} \begin{bmatrix} \Delta X \\ \Delta Y \\ \Delta S \end{bmatrix} = \begin{bmatrix} -r_C \\ -r_B \\ -X^d S^d e - (\Delta X^d)^{aff}(\Delta S^d)^{aff} e + \sigma\mu e \end{bmatrix}. \tag{3.33}$$

Análogamente a lo que se hizo anteriormente para transformar el sistema 3.15 en un sistema de ecuaciones equivalente conformado por las ecuaciones 3.23 y 3.24, se obtiene lo siguiente ($D$ sigue teniendo la definición anterior):

$$\begin{bmatrix} 0 & A \\ A^T & -D^{-2} \end{bmatrix} \begin{bmatrix} \Delta Y \\ \Delta X \end{bmatrix} = \begin{bmatrix} -r_B \\ -r_C + S + (X^d)^{-1}(\Delta X^d)^{aff}(\Delta S^d)^{aff} e - \sigma\mu(X^d)^{-1} e \end{bmatrix},$$
$$\Delta S = -S - (X^d)^{-1} S^d \Delta X - (X^d)^{-1}(\Delta X^d)^{aff}(\Delta S^d)^{aff} e + \sigma\mu(X^d)^{-1} e.$$

También se obtiene un sistema equivalente al de la ecuación 3.28, en el cual el paso costoso es la factorización de Cholesky de la matriz $AD^2A^T$:

$$\begin{aligned} AD^2A^T\Delta Y &= -r_B + A(-(S^d)^{-1}X^d r_C + X + (S^d)^{-1}(\Delta X^d)^{aff}(\Delta S^d)^{aff} e - \sigma\mu(S^d)^{-1} e), \\ \Delta S &= -r_C - A^T\Delta Y, \\ \Delta X &= -X - (S^d)^{-1}X^d\Delta S - (S^d)^{-1}(\Delta X^d)^{aff}(\Delta S^d)^{aff} e + \sigma\mu(S^d)^{-1} e. \end{aligned} \tag{3.34}$$

Los pasos más grandes que pueden darse en estas direcciones sin violar las condiciones de no negatividad $(X, S) > 0$ se calculan análogamente a como se calcularon $\alpha_{aff}^{pri}$ y $\alpha_{aff}^{dual}$ anteriormente:

$$\alpha_{max}^{pri} = \text{mín}\left(1, \min_{i:\Delta x_i<0} -\frac{x_i^k}{\Delta x_i}\right), \tag{3.35}$$

$$\alpha_{max}^{dual} = \text{mín}\left(1, \min_{i:\Delta s_i<0} -\frac{s_i^k}{\Delta s_i}\right). \tag{3.36}$$

Las longitudes de paso para las variables primales y duales, respectivamente, se escogen de la siguiente forma:

$$\alpha_k^{pri} = \text{mín}(1, \eta\alpha_{max}^{pri}), \tag{3.37}$$
$$\alpha_k^{dual} = \text{mín}(1, \eta\alpha_{max}^{dual}). \tag{3.38}$$

El valor $\eta \in [0.9, 1.0)$ se escoge tal que $\eta \to 1$ cerca de la solución, para de esta forma acelerar la convergencia asintótica. Se puede escoger $\eta = 0.99$ según [Wri97]. Como criterios de parada, se pueden evaluar la factibilidad primal y dual, entre otras condiciones.

[8]Los componentes del predictor, el centrado y del corrector se obtienen resolviendo sistemas de ecuaciones lineales con la misma matriz de coeficientes, y como son independientes el uno del otro, no es necesario calcularlos separadamente. Se combinan en una única dirección añadiendo los vectores correspondientes a la derecha de las ecuaciones.

**Algoritmo de Mehrotra**

Escoger el punto inicial $(X^0, Y^0, S^0)$, tal que $(X^0, S^0) > 0$[9].

**for** $k = 0, 1, \ldots,$

$(X, Y, S) = (X^0, Y^0, S^0)$;

Resolver 3.15 para $(\Delta X^{aff}, \Delta Y^{aff}, \Delta s^{aff})$;

Calcular $\alpha_{aff}^{pri}$, $\alpha_{aff}^{dual}$ y $\mu_{aff}$ como en 3.29 y 3.30;

Calcular el parámetro de centrado $\sigma$ según lo señalado en 3.32;

Resolver 3.33 para $(\Delta X, \Delta Y, \Delta S)$;

Calcular $\alpha_k^{pri}$ y $\alpha_k^{dual}$ como en 3.35 y 3.36;

Hacer: $X^{k+1} = X^k + \alpha_k^{pri} \Delta X$;

Hacer: $Y^{k+1} = Y^k + \alpha_k^{dual} \Delta Y$;

Hacer: $S^{k+1} = S^k + \alpha_k^{dual} \Delta S$;

**end for**.

## 3.5. Programación Entera

La programación entera (PE) es un término que se utiliza para los modelos de programación matemática cuyas variables (todas o solamente algunas) están condicionadas a tomar valores enteros. En muchos campos de la vida real, surgen problemas como éstos, donde las variables de decisión son inherentemente enteras, debido a la naturaleza del problema de toma de decisión. Se debe entonces considerar el siguiente problema de optimización lineal:

$$\begin{aligned} &\min_{x_1,\ldots,x_n} C^T X \\ &s.a.\ AX = B, \\ &\quad x_i \geq 0 \quad \forall i \in [1, n], \qquad\qquad x_i \in \mathbb{Z}, \text{para todos o algunos } i \in [0, n]. \end{aligned}$$

Cuando todas las variables están restringidas a tomar valores enteros, es un problema puro de programación entera. Si, por el contrario, sólo algunas variables están condicionadas a la integridad, se está tratando con un problema mixto de programación entera. Si las variables deben tomar valores de ceros o unos solamente, entonces se trata de un problema de programación entera binaria.

### 3.5.1. Relajación Lineal de un PE

Dado un problema de programación entera (IP), existe un problema de programación lineal asociado llamado **problema relajado** (*linear relaxation*) (LR). Este

[9]Para más detalles, véase [Meh92]-pág 589.

problema es el mismo (IP) salvo que las restricciones de integridad son eliminadas. Sean $Opt(IP)$ y $Opt(LR)$ los valores de la función objetivo evaluada en la solución óptima de (IP) y (LR), respectivamente. Como el problema (LR) es menos restrictivo que (IP), se cumple lo siguiente [BK05]:

1. Si (IP) es un problema de minimización, entonces $Opt(LR) \leq Opt(IP)$.

2. Viceversa, si (IP) es un problema de maximización, entonces $Opt(LR) \geq Opt(IP)$.

3. Si (LR) no tiene solución, entonces (IP) tampoco la tiene.

4. Si todas las variables en una solución óptima de (LR) tienen valores enteros, entonces ésa solución es óptima para (IP) también.

5. Si los coeficientes de la función objetivo son enteros, entonces para los problemas de minimización, $Opt(IP) \geq \lceil Opt(LR) \rceil$. Para problemas de maximización, $Opt(IP) \leq \lfloor Opt(LR) \rfloor$.

Por lo tanto, resolver el problema relajado (LR) puede ser útil, pues provee una cota para el valor de la función objetivo de (IP) y pudiera obtenerse la solución óptima para (IP) si la solución óptima resulta ser entera.

### 3.5.2. Ramificación y poda

La técnica conocida como Ramificación y Poda (*branch and bound*) ha sido ampliamente utilizada para resolver los problemas de PE. Este procedimiento se basa en la división del problema en subproblemas (ramificación) mientras el óptimo no sea una solución factible entera.

La Ramificación y Poda es un método general de búsqueda. Comienza considerando el problema original con la región factible completa sin las restricciones de integridad, es decir, considerando el problema relajado. Si la solución óptima encontrada es entera, el procedimiento ha terminado. En caso contrario, la región de factibilidad se divide en dos o más regiones (ramificación). Se resuelve cada uno de los subproblemas, y si uno de ellos satisface las restricciones de integridad, el problema ha sido resuelto pero no se puede garantizar su optimalidad. Si no satisface dichas restricciones, el valor de la función objetivo aporta una cota superior o inferior del óptimo, según sea el problema de maximización o minimización. Si el valor de dicha cota es peor que la mejor cota obtenido hasta el momento, no hará falta seguir explorando este problema (poda). Este procedimiento se resuelve de modo iterativo para cada uno de los subproblemas generados. Para más información sobre cómo seleccionar la variable a partir de la cual se hará la ramificación, y cuál subproblema resolver primero, véase [HL05].

A continuación a modo de ejemplo, considérese el siguiente problema mixto de PE:

$$\begin{aligned} \text{máx } & 7x_1 + 3x_2 + 4x_3, \\ s.a.\ & 3x_1 + 2x_2 - x_3 \leq 5, \\ & 7x_1 + 2x_2 \leq 8, \\ & 2x_2 - 3x_3 \leq 9, \\ & x_1, x_3 \in \mathbb{Z}^+, x_2 \geq 0. \end{aligned} \tag{3.39}$$

La solución del LR es $(1.1428, 0, 3)$ y el valor de la función objetivo evaluada en este vector es $z = 14.2857$. Como la solución no es entera, se deben generar dos nuevos subproblemas mediante la adición de las restricciones $x_1 \leq 1$ y $x_1 \geq 2$ al problema original 3.39, una para cada subproblema. En la Figura 3.4 se muestra el método de ramificación y poda aplicado a este ejemplo.

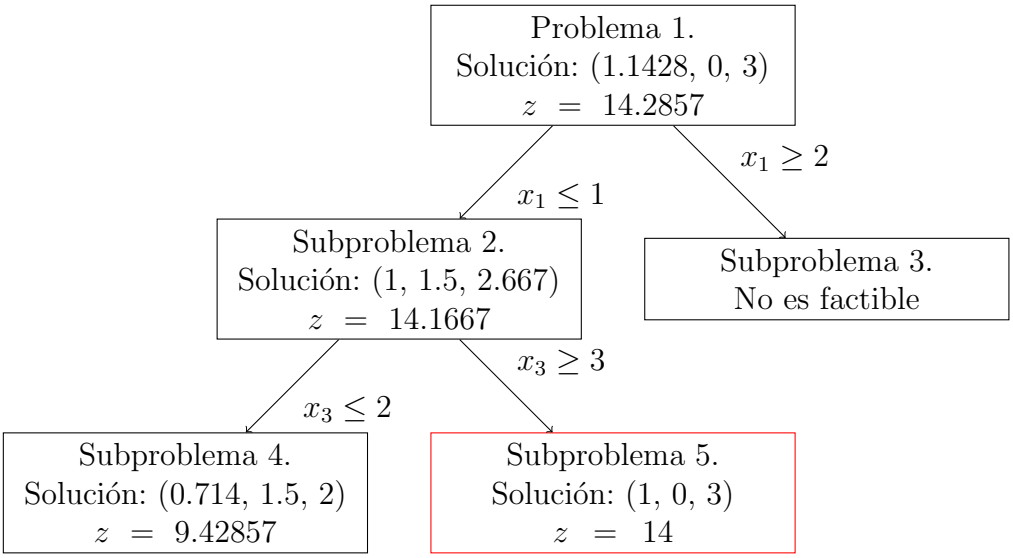


Figura 3.4: Ramificación y poda para el ejemplo 3.39.

La solución del LR del segundo subproblema es $(1, 1.5, 2.667)$ y el valor de la función objetivo evaluada en esta solución es $z = 14.166$. Esta es una cota superior más ajustada, aunque la solución aún no es factible puesto que $x_3 \notin \mathbb{Z}^+$. Por otra parte, la relajación del tercer subproblema no tiene solución, por lo que esa rama se poda. El próximo paso es ramificar el segundo subproblema añadiéndole las restricciones $x_3 \leq 2$ y $x_3 \geq 3$. La relajación del cuarto subproblema es $(0.714, 1.5, 2)$ y el valor de la función objetivo evaluada en esta solución es $z = 9.42857$, mientras que la relajación del quinto subproblema es $(1, 0, 3)$, para la cual $z = 14$. Como esta última solución es entera, y además la cota superior que provee el cuarto subproblema es inferior a la del quinto subproblema, se concluye que $(1, 0, 3)$ es la solución óptima del problema original.

### 3.5.3. Planos cortantes

Los métodos del plano cortante (*cutting planes*) se utilizan para resolver problemas de programación entera. El método consiste en relajar los problemas enteros a problemas de Programación Lineal. Si la solución resultante del problema es entera, el problema ha sido resuelto. En caso contrario, se plantea una nueva restricción (plano de corte), que se añadirá al problema. Estas nuevas restricciones pretenden "relajar"las aristas que dan soluciones no enteras; es decir, eliminar las soluciones fraccionarias, sin eliminar ningúna solución entera. El nuevo problema se vuelve a resolver como problema de Programación Lineal y el proceso anterior se repite hasta que la solución es entera. Aunque se ha demostrado que el número de cortes necesarios para garantizar la optimalidad es finito, en realidad la convergencia es un poco lenta [HL05].

A continuación se muestra el fundamento geométrico del método para el siguiente ejemplo sencillo:

$$\begin{aligned} &\text{máx } 7x_1 + 10x_2, \\ &s.a.\ 7x_1 + x_2 \leq 35, \\ &\quad -x_1 + 3x_2 \leq 6, \\ &\quad x_1, x_2 \geq 0, \\ &\quad x_1, x_2 \in \mathbb{Z}. \end{aligned} \tag{3.40}$$

Las siguientes figuras muestran el procedimiento iterativo de los planos de corte para el ejemplo anterior. Con dos cortes, se alcanza la solución óptima entera que es $(4; 3)$:

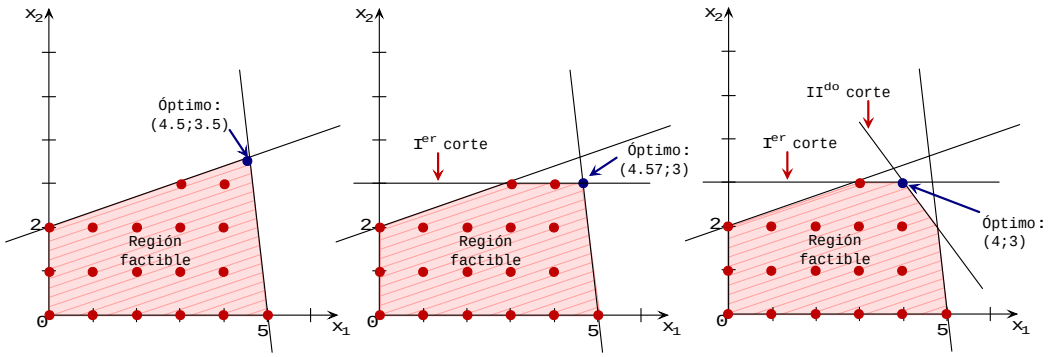


Figura 3.5: Idea del método de los planos cortantes para el ejemplo 3.40.

### 3.5.4. Algoritmo de Balas

Los problemas de programación lineal entera binaria pueden ser resueltos usando cualquiera de las técnicas generales de programación lineal como el método de planos cortantes de Gomory [Gom58], el método de ramificación y poda de Land y Doig

(*branch & bound*) [AHL60] (introduciendo la restricción adicional de que todas las variables tienen que tomar valores de ceros o unos), la programación dinámica y la relajación Lagrangiana. No obstante, los métodos que han sido desarrollados originalmente para resolver problemas generales de programación lineal entera, no aprovechan las características especiales de los problemas de programación lineal binaria. Para resolver estos problemas más específicos de manera más eficiente, han sido propuestos varios métodos. Uno de ellos es el algoritmo de Balas.

El algoritmo aditivo de Balas fue formulado en 1965 para resolver un problema puro de programación lineal entera binaria [Bal65]. En este algoritmo, todas las posibles soluciones ($2^n$) son enumeradas ya sea explícita o implícitamente. La eficiencia del método estriba en la estrategia que adopta, de seleccionar sólo unas pocas soluciones para enumeración explícita. El método comienza asignando a todas las variables el va- lor de cero, y consiste en un procedimiento sistemático de asignaciones sucesivas a ciertas variables del valor 1, de tal forma que luego de tratar una pequeña parte de todas las posibles combinaciones, obtiene o una solución óptima, o una evidencia del hecho que no existe tal solución óptima. Básicamente, es un enfoque de ramificación y poda, pero aplicado al problema específico de la programación entera. Este enfoque se denomina enumeración implícita. La palabra implícita significa que, aunque no se enumeren todas las soluciones, no es posible que se ignore una posible solución óptima por causa de la exclusión de otras soluciones.

El algoritmo de Balas no es eficiente en términos de requerimientos de almacenamiento [Pet67, Geo67], debido a la forma en que se lleva el registro de las soluciones que han sido ya enumeradas y/ó excluidas. Es por eso que aquí se ha decidido emplear la reformulación de Geoffrion del algoritmo aditivo de Balas [Geo67].

# Capítulo 4

# Enfoque de Programación Entera Binaria al problema de la Asignación

## 4.1. Modelo de Tareas

En este trabajo de tesis, se tratará el problema de planificar un conjunto $\tau = \{\tau 1, \ldots, \tau_n\}$ de $n$ tareas en un número fijo de $m$ procesadores de características idénticas. Se considerará un sistema de tiempo real crítico, es decir, que todos los plazos deben cumplirse a fuerzas, y la planificación se realizará fuera de línea. Las tareas no comparten recursos y no tienen relaciones de dependencia. Además, se conocen *a priori* los siguientes parámetros que caracterizan al conjunto de tareas:

- $C_i$: el tiempo de ejecución de la tarea $\tau_i$,
- $T_i$: el período de la tarea $\tau_i$, y
- $u_i = \frac{C_i}{T_i}$: la utilización de la tarea $\tau_i$.

Además, la planificación se realizará bajo el esquema particionado, de forma tal que en cada procesador las tareas asignadas se planificarán bajo el algoritmo EDF. La prueba de planificabilidad que se usará será la condición necesaria y suficiente de Liu y Layland para EDF del Teorema 2.10.

Se considerará que no se puede dividir ninguna tarea en partes y que no habrá migración a nivel de instancias de tarea, o sea, que cada tarea debe asignarse en su totalidad a un sólo procesador (y todas sus instancias deberán ejecutarse en él), o a ninguno, en el peor caso.

## 4.2. Modelo Matemático

Denotemos por:

$$X_{ij} = \begin{cases} 1 \text{ si la tarea } \tau_i \text{ es asignada al procesador } j, \\ 0 \text{ en caso contrario.} \end{cases} \tag{4.1}$$

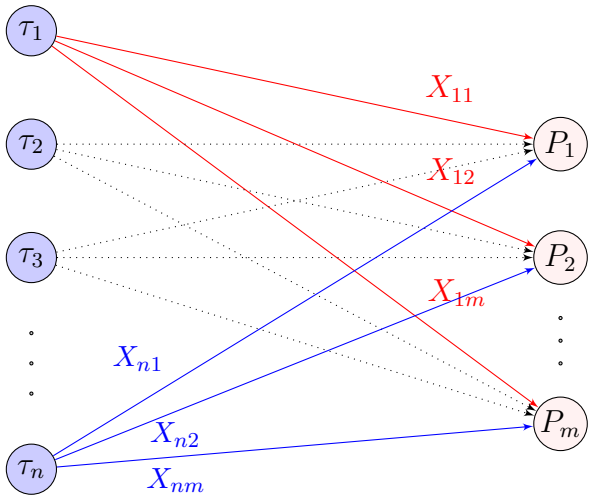


Figura 4.1: Asignación de las $n$ tareas a los $m$ procesadores.

La Figura 4.1 puede esclarecer la formación de la función objetivo y las restricciones para el modelo matemático de la planificación multiprocesador bajo el modelo de tareas especificado en la sección anterior. El objetivo aquí es maximizar la utilización de los procesadores. Cuando la utilización total no sobrepasa la capacidad total disponible, esto es equivalente a maximizar el número de tareas a asignar a los procesadores. Esto es, se necesita maximizar la siguiente función:

$$\sum_{i=1}^{n} u_i \Big( \sum_{j=1}^{m} X_{ij} \Big). \tag{4.1}$$

Además, cada tarea puede ser asignada solamente a un procesador o a ninguno, puesto que no se pueden dividir en partes. Esto es, para cada $i \in [1, n]$ se tiene que se debe cumplir la siguiente ecuación:

$$\sum_{j=1}^{m} X_{ij} \leq 1. \tag{4.2}$$

En cada procesador, para que las tareas asignadas puedan ser planificables bajo EDF, se debe cumplir la prueba necesaria y suficiente de planificabilidad dada por Liu y

Layland del Teorema 2.10. Esto significa que para cada $j \in [1, m]$, se debe cumplir que las utilizaciones de las tareas asignadas al procesador $j$-ésimo no sobrepasen 1:

$$\sum_{i=1}^{n} u_i X_{ij} \leq 1. \tag{4.3}$$

Entonces, el problema de la asignación puede modelarse como el siguiente problema de optimización lineal entera binaria con $m \times n$ variables binarias y $m+n$ restricciones:

$$\text{máx} \sum_{i=1}^{n} u_i \Big( \sum_{j=1}^{m} X_{ij} \Big) \tag{4.4}$$

$$\text{s.a.} \sum_{i=1}^{n} u_i X_{ij} \leq 1, \quad j = 1, \ldots, m, \tag{4.4a}$$

$$\sum_{j=1}^{m} X_{ij} \leq 1, \quad i = 1, \ldots, n, \tag{4.4b}$$

$$X_{ij} \in \{0, 1\}, \quad i = 1, \ldots, n, \; j = 1, \ldots, m. \tag{4.4c}$$

## 4.3. Algoritmo propuesto

Proponemos que aquí se aplique la versión de Geoffrion del Algoritmo de Balas dada en [Geo67], conjuntamente con algunas modificaciones que se le realizaron, basadas en los conocimientos del problema específico de la planificación multiprocesador. Estas modificaciones se detallarán en la Sección 4.4.

El algoritmo de Geoffrion-Balas [Geo67] se aplica a un problema puro de programación lineal entera binaria, especificado en la forma estándar:

$$\text{mín}\, C^T X \tag{4.5}$$

$$s.a. \; B + AX \geq 0, \tag{4.14a}$$

$$x_i = 0 \text{ ó } 1 \; \forall i \in [0, n], \tag{4.14b}$$

donde $C$ es un vector $n$-dimensional, $B$ es un vector $m$-dimensional, $A$ es una matriz de tamaño $m \times n$ y $X$ es un vector de variables binarias. Cualquier $X$ binario se considerará una posible solución del problema anterior. Si esta solución $X$ además satisface las restricciones de desigualdad, entonces es una solución **factible**. Una solución factible que minimiza el valor de la función objetivo $C^T X$ sobre todas las demás soluciones factibles, se denomina una **solución factible óptima**.

Para reducir el problema de optimización lineal entera binaria 4.4 a la forma estándar que se necesita para aplicar la reformulación del algoritmo de Balas dada por Geoffrion [Geo67], se necesita el siguiente cambio de variables:

$$Y_{ij} = 1 - X_{ij}, \;\; i = 1, \ldots, n, \; j = 1, \ldots, m. \tag{4.6}$$

De manera que la función objetivo se convierte en:

$$\text{mín} - \sum_i \sum_j (1 - Y_{ij}) = \text{mín} \sum_i \sum_j (u_i Y_{ij} - u_i) \equiv \text{mín} \sum_i \sum_j (u_i Y_{ij}) \tag{4.7}$$

Sustituyendo 4.6 en 4.2, se tiene:

$$(1 - Y_{i1}) + (1 - Y_{i2}) + \ldots + (1 - Y_{im}) \leq 1, \ \forall i \in [1, n]. \tag{4.8}$$

Es decir, se obtienen las $n$ restricciones siguientes:

$$Y_{i1} + \ldots + Y_{im} + (1 - m) \geq 0, \ \forall i \in [1, n]. \tag{4.9}$$

Sustituyendo 4.6 en 4.3, se obtiene:

$$u_1(1 - Y_{1j}) + u_2(1 - Y_{2j}) + \ldots + u_n(1 - Y_{nj}) \leq 1, \ \forall j \in [1, m]. \tag{4.10}$$

De manera que las $m$ restricciones relativas a la prueba exacta de planificabilidad para EDF son:

$$u_1 Y_{1j} + \ldots + u_n Y_{nj} + (1 - \sum_{i=1}^{n} u_i) \geq 0, \ \forall j \in [1, m]. \tag{4.11}$$

Así, se tiene el problema en la forma estándar que se requiere para aplicar el algoritmo de Geoffrion, con las variables $Y_{ij}$ binarias:

$$\text{mín} \sum_i \sum_j u_i Y_{ij} \tag{4.12}$$

$$s.a. \sum_{j=1}^{m} Y_{ij} + (1 - m) \geq 0, \quad i = 1, \ldots, n, \tag{4.6a}$$

$$\sum_{i=1}^{n} u_i Y_{ij} + (1 - \sum_{i=1}^{n} u_i) \geq 0, \quad j = 1, \ldots, m. \tag{4.6b}$$

El $(1) \times (n \times m)$ vector $B^T$ es:

$$(\underbrace{1 - m, \ldots, 1 - m}_{n \text{ veces}}, \underbrace{1 - \sum_{i=1}^{n} u_i, \ldots, 1 - \sum_{i=1}^{n} u_i}_{m \text{ veces}}). \tag{4.13}$$

Y la $(n + m) \times (n \times m)$ matriz $A$ tiene la siguiente forma:

$$\begin{pmatrix}
1 & 1 & \ldots & 1 & 0 & 0 & \ldots & 0 & \ldots & 0 & 0 & \ldots & 0 \\
0 & 0 & \ldots & 0 & 1 & 1 & \ldots & 1 & \ldots & 0 & 0 & \ldots & 0 \\
\vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots & \ldots & \vdots & \vdots & \ddots & \vdots \\
0 & 0 & \ldots & 0 & 0 & 0 & \ldots & 0 & \ldots & 0 & 0 & \ldots & 0 \\
0 & 0 & \ldots & 0 & 0 & 0 & \ldots & 0 & \ldots & 1 & 1 & \ldots & 1 \\
u_1 & 0 & \ldots & 0 & u_2 & 0 & \ldots & 0 & \ldots & u_n & 0 & \ldots & 0 \\
0 & u_1 & \ldots & 0 & 0 & u_2 & \ldots & 0 & \ldots & 0 & u_n & \ldots & 0 \\
\vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots & \ldots & \vdots & \vdots & \ddots & \vdots \\
0 & 0 & \ldots & u_1 & 0 & 0 & \ldots & u_2 & \ldots & 0 & 0 & \ldots & u_n
\end{pmatrix}. \tag{4.14}$$

### 4.3.1. Reformulación de Geoffrion

Las principales ventajas de la reformulación de Geoffrion del algoritmo de Balas son las que se citan a continuación:

1. El algoritmo requiere una cantidad considerablemente menor de memoria de almacenamiento que el algoritmo de Balas original.

2. La única operación matemática que se realiza es la adición, lo cual evita los errores de redondeo y los manejos de operaciones con matrices.

3. Usualmente se tiene una solución factible almacenada (aunque quizás no la óptima) si por alguna razón los cálculos deben ser interrumpidos antes de llegar a término. De hecho, la solución disponible cuando sean detenidos los cálculos puede ser óptima (o llevar a la óptima): el tiempo requerido para el resto de los cálculos puede ser suficiente para probar la optimalidad de la solución por medio de la enumeración exhaustiva del resto de las soluciones posibles.

4. Es posible monitorear cuántas soluciones han sido implícitamente enumeradas.

5. Existen numerosos modos de particularizar el algoritmo, empleando conocimiento previo relativo a problemas específicos.

6. También se pueden manejar funciones objetivo no lineales.

**Definición 4.3.1.** *Sea $V \subset \{1, \ldots, n\}$. Una solución parcial $S$ para el problema 4.5 es una asignación de valores binarios a un subconjunto de las $n$ variables:*

$$S = \{k \text{ ó } -k \mid k \in V\}, \tag{4.15}$$

*donde $k$ denota que $x_k = 1$, y $-k$ denota que $x_k = 0$.*

**Definición 4.3.2.** *Las variables libres en la solución parcial $S$ son las que no tienen valores asignados, o sea, pertenecen a $V^C = \{k \in \{1, \ldots, n\} \mid k \notin V\}$.*

**Definición 4.3.3.** *Un completo de la solución parcial $S$ consiste en una solución que está determinada por $S$ conjuntamente con una especificación de valores binarios de las variables libres:*

$$S \cup \{k \text{ ó } -k \mid \forall k \in V^C\}. \tag{4.16}$$

El conjunto de completos de $S$ (denotado por $SC$) está determinado por todos los posibles conjuntos de la forma anterior. De esta manera, una solución parcial $S$ con $p$ elementos, determina un conjunto de $2^{n-p}$ posibles completos. Véase la Figura 4.2 para una representación de las soluciones parciales y sus completos en un problema con cinco variables binarias.

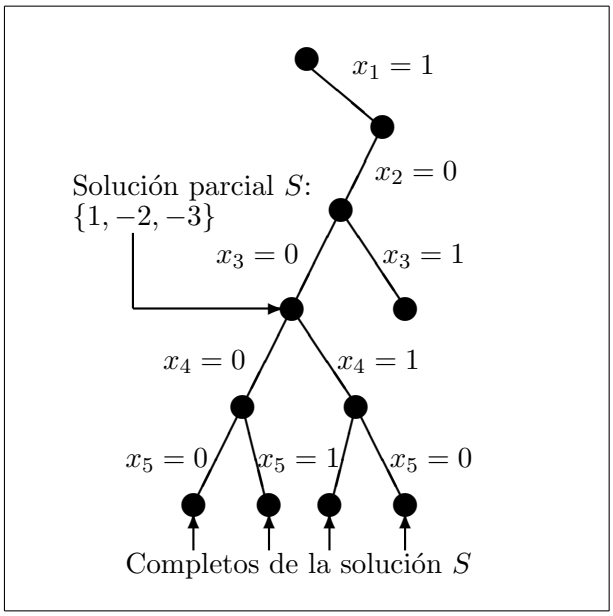


Figura 4.2: Soluciones parciales y completos en un árbol de enumeración.

**Definición 4.3.4.** *El mejor completo factible de la solución parcial $S$ es aquel que minimiza el valor de la función objetivo entre todos los posibles completos factibles de $S$:*

$$\min_{X \in SC, AX+B \geq 0} C^T X. \tag{4.17}$$

**Definición 4.3.5.** *Se dice que la solución parcial $S$ está sondeada si ocurre una de las dos situaciones siguientes:*

1. *Si se puede determinar el mejor completo de $S$ y éste es mejor que la mejor solución que se conoce hasta el momento, ó*

2. *si se determina que $S$ no tiene ningún completo factible mejor que la mejor solución que se conoce hasta el momento.*

El método genera una sucesión de soluciones parciales y considera, simultáneamente, todos sus completos. A medida que se desarrollan los cálculos, se encuentran soluciones factibles de cuando en cuando, almacenándose siempre la que provee un menor valor de la función objetivo.

**Teorema 4.3.1.** *[Geo67] El procedimiento de enumeración implícita de la Figura 4.3 conlleva a una sucesión no redundante de soluciones parciales, y termina solamente cuando todas las $2^n$ soluciones han sido implícitamente enumeradas.*

La idea del algoritmo de Geoffrion-Balas radica en moverse por el árbol de soluciones, tratando de sondear cada solución parcial (poda). Si una solución parcial no se puede sondear, se desciende aún más por una de las dos ramas del árbol que parten de

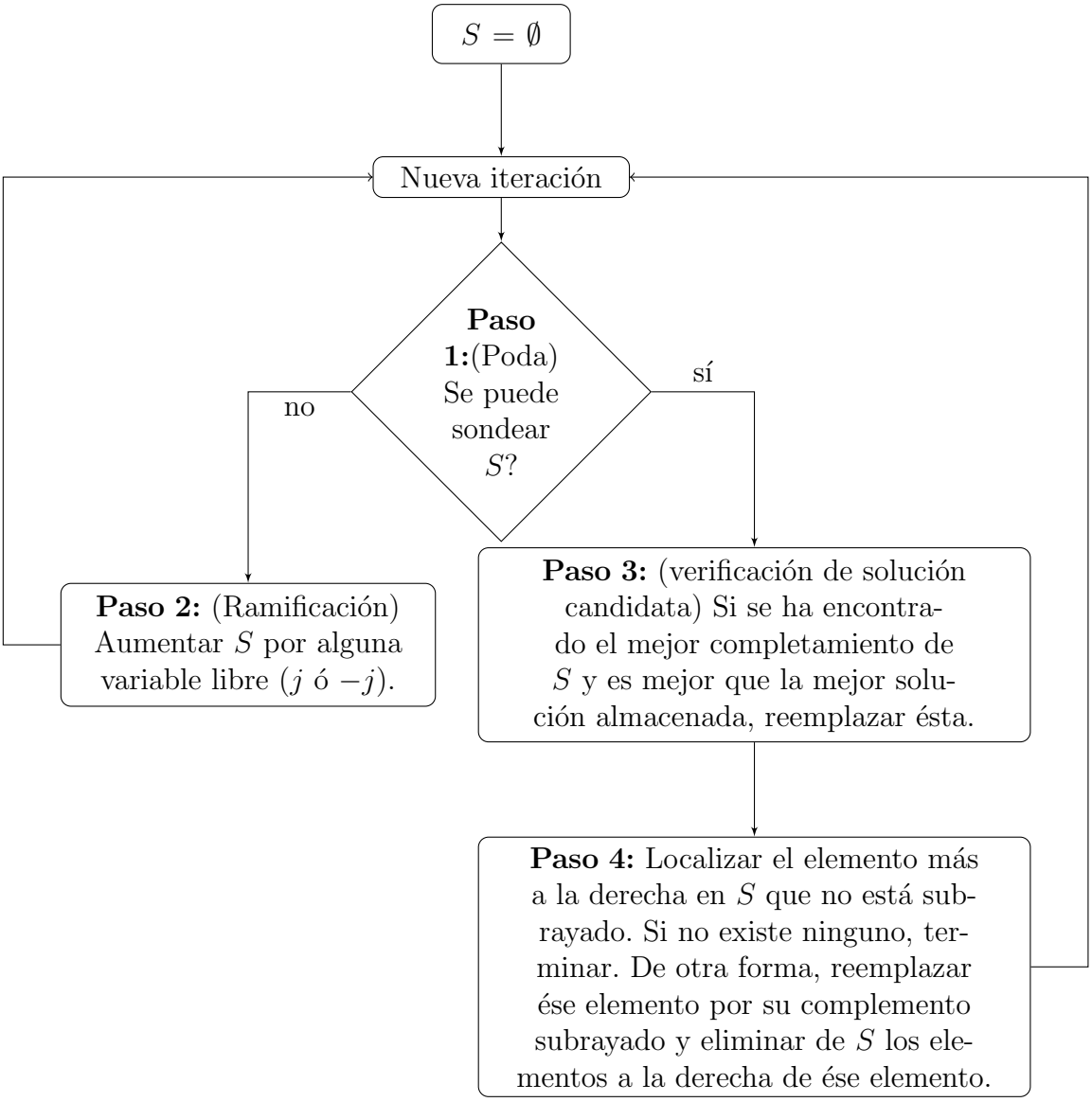


Figura 4.3: Procedimiento de *backtracking* para enumeración implícita.

esa solución parcial (ramificación), hasta que eventualmente, un nodo esté sondeado. Cuando ésto sucede, se toma la otra rama del nodo padre al nodo sondeado y se continúa la búsqueda. Este es un algoritmo de ramificación y poda específico para el caso de variables binarias.

Como se ve, es un procedimiento de vuelta atrás (*backtracking*). Ésta es una técnica muy conocida de programación para hacer una búsqueda a través de todas las configuraciones posibles dentro de un espacio de búsqueda. En los algoritmos de este tipo, sistemáticamente se construyen soluciones candidatas para abandonarlas, tan pronto se determine que éstas no pueden ser completadas para formar una solución válida

del problema.

En el paso 1 de la Figura 4.3, se debe tratar de sondear la solución parcial $S$, hallando su mejor completo factible, o bien, determinando que ningún completo factible de $S$ es mejor que la mejor solución que se tiene almacenada hasta el momento. Pero es muy difícil hallar el mejor completo factible de una solución parcial.

El completo trivial de una solución parcial es aquel en el que todas las variables libres toman el valor de cero. Si el completo trivial resulta factible, éste es el mejor completo de $S$. Puesto que se está minimizando la función objetivo, es éste completo el que provee el menor valor de $CX$, pero no se asegura que sea factible (que cumpla con las restricciones del problema). Lo que se hará aquí es evaluar la factibilidad de ese completo trivial, y en caso que no resulte factible (porque algún elemento del vector $Y^S = AX^S + B$ fue negativo), no se hará nada más para encontrar un mejor completo factible de $S$ (este es el paso 1a de la Figura 4.5). En cambio, se tratará de determinar que ningún completo de $S$ es mejor que la mejor solución almacenada. En este caso, será imposible completar $S$ dándole valores a las variables libres de forma tal que se eliminen todas las infactibilidades de $X^S$, y aún mejorar la cota $\bar{z}$.

Para demostrar que $S$ es imposible de completar en esta forma, se deben contemplar las variables libres a las que se le pueden dar el valor de uno, tal que su coeficiente asociado en la matriz de restricciones sea distinto de cero, y tal que mejoren la cota de la función objetivo. Si el coeficiente asociado a una variable libre en la fila $i$-ésima de la matriz de restricciones es distinto de cero, al darle el valor de uno a esta variable, se estará tratando de eliminar la infactibilidad dada por el valor negativo del elemento $i$-ésimo del vector $Y^S$. Si también se mejora la cota de la función objetivo al darle el valor de uno, ésta variable libre será una posible candidata para agregarla a $S$ en el paso 2 (Figura 4.3).

Entonces, se debe calcular el conjunto que se denotará por $T^S$, que es el conjunto de variables libres que pudiesen eliminar las infactibilidades de las restricciones, y al mismo tiempo, mejorar la cota de la función objetivo:

$$T^S = \{j \in V^C \mid CX^S + c_j < \bar{z} \text{ y } A_{ij} > 0,\ i \text{ es tal que } Y_i^S < 0\}. \tag{4.18}$$

Al darle el valor de uno a alguna variable libre que no esté en el conjunto $T^S$, se obtendría un mayor valor de la cota $\bar{z}$, o bien, no contribuiría a disminuir una infactibilidad en $Y^S$. Por lo tanto, si $T^S$ resulta vacío, no existe ningún completo de $S$ mejor que la mejor solución almacenada, y $S$ estaría sondeado. De igual forma, $S$ estaría sondeado si para algún $i$ tal que $Y_i^S < 0$, se cumple:

$$Y_i^S + \sum_{j \in T^S} A_{ij} < 0, \tag{4.19}$$

pues no habría forma de seleccionar variables libres para eliminar las infactibilidades de las restricciones (valores negativos de los elementos de $Y^S$). Véase en la ecuación

anterior, que, si por más que se le puedan sumar coeficientes a $Y_i^S$, no se elimina la infactibilidad, entonces no tiene sentido seguir añadiendo valores a $S$ y por tanto, se poda la rama.

Resumiendo, los criterios de sondeo (ó poda) son los siguientes:

- Si $Y^S \geq 0$, pues se habrá encontrado el mejor completo de $S$.
- Si $T^S = \emptyset$ ó se cumple la Ecuación 4.19 para algún $i$ tal que $Y_i^S < 0$, pues no habría forma de eliminar las infactibilidades de las restricciones.

En el paso 2, correspondiente a la ramificación, (Figura 4.3), se deben añadir a $S$ una o más variables libres, de forma tal que se disminuya lo más que se pueda la infactibilidad de la solución parcial. Una opción es añadir la variable libre $j_0 \in T^S$ (con valor de uno) de forma tal que la siguiente expresión sea un máximo algebraico:

$$\sum_{i=1}^{m+n} \text{mín}\{Y_i^S + A_{ij}, 0\}. \tag{4.20}$$

Para llevar la cuenta de los sondeos, se introduce una notación de subrayados como se explicará a continuación:

Supóngase que, en la iteración $k$-ésima, $S^k$ ha sido sondeado. De acuerdo al algoritmo de la figura 4.3, se procede entonces al paso 4 (vuelta atrás). Debe almacenarse el conjunto $S^k$, para poder tener suficiente información sobre cuántas soluciones han sido explícitamente enumeradas. El método genera una sucesión de soluciones parciales no redundantes porque cada $S^{k+1}$ se construye de forma tal que contenga al menos un elemento complementario a alguno de los que están en $S^k$. $S^{k+1}$ se toma como $S^k$, con su último elemento multiplicado por -1 y subrayado. O sea, si la última variable en $S^k$ tenía valor de uno, en $S^{k+1}$ tendrá el valor de cero y viceversa.

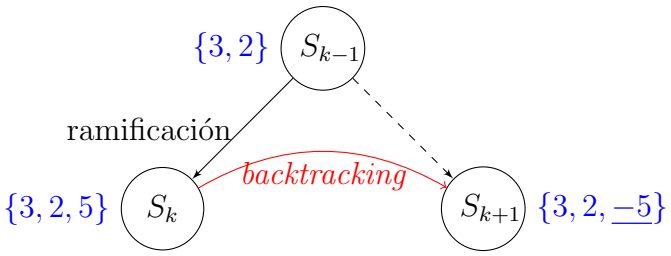


Figura 4.4: Tres soluciones parciales consecutivas.

En la figura 4.4 se representan tres soluciones parciales consecutivas resultantes del algoritmo. Lo que hace el método aplicado al ejemplo de la Figura 4.4 es añadir la variable 5 con valor de uno al no poder sondear $S_{k-1}$. Después, se pasa a una nueva iteración en la que se puede sondear $S_k$; y en el paso 4 (*backtracking*) de la figura 4.3, se pasa a $S_{k+1}$, que es el otro nodo hijo de $S_{k-1}$. Cuando $S_{k+1}$ haya sido sondeado,

habrá sido sondeado a su vez $S_{k-1}$, puesto que los completamientos de $S_k$ y de $S_{k+1}$ dicotomizan los de $S_{k-1}$.

Si inmediatamente $S_{k+1}$ no puede ser sondeado, se le añade una ó más variables libres a esa solución parcial, cada vez tratando de sondearla, hasta que, eventualmente, ésto suceda en la iteración $p$-ésima. Nótese que la sucesión de soluciones candidatas $S_{k+1}, \ldots, S_p$ es no redundante pues cada una contiene el complemento de un elemento de $S_k$.

Cuando $S_p$ haya sido sondeado, debe ser almacenado junto con la solución $S_k$, y todas las soluciones candidatas posteriores a $S_p$ deben contener, no sólo el complemento de un elemento en $S_k$, sino también el complemento de un elemento de $S_p$. Esto puede lograrse tomando $S_{p+1}$ como $S_p$ con su último elemento multiplicado por menos uno y subrayado. Se puede llevar el registro de este historial de sondeos almacenando en un mismo conjunto estos cambios de signo y de subrayados, para evitar almacenamiento de más, como hace el algoritmo original de Balas.

Es por esto que en el paso 4 de la Figura 4.3 se busca el último elemento que no está subrayado, se sustituye por su complemento subrayado y se eliminan los demás elementos a su derecha (se salta al último nodo hermano a un nodo sondeado).

## 4.4. Modificaciones al algoritmo

La versión de Geoffrion fue adaptada con la introducción de tres modificaciones basadas en las características del problema de la asignación multiprocesador con el que se está tratando y que son las siguientes:

1. **Comenzar con una solución parcial no nula**. Antes de entrar al ciclo de la Figura 4.3, se forma una solución inicial de la siguiente forma: las tareas se ordenan de forma decreciente según sus utilizaciones, y las primeras $m$ tareas son asignadas a los $m$ procesadores. En la práctica se ha apreciado que las heurísticas bin-packing funcionan mejor en los conjuntos de tareas que han sido ordenados de forma decreciente según sus utilizaciones [PZMA04]. Por consiguiente, la solución inicial es tal que la tarea con mayor utilización es asignada al primer procesador, la tarea con segunda mayor utilización es asignada al segundo procesador, y así sucesivamente. Nuestros resultados experimentales mostraron que en general, la solución inicial forma parte de la solución óptima. Esto implica una reducción en el número de iteraciones para encontrar la solución óptima.

2. **Verificar la planificación cada vez que se encuentre una nueva cota para la función objetivo, y terminar si todas las tareas han sido planificadas**. En el paso 3 de la Figura 4.3, el algoritmo evalúa todas las posibles soluciones buscando una mejor cota de la función objetivo. Sin embargo, debido a la naturaleza del problema específico de la asignación, una vez que todas las

tareas han sido asignadas, no es necesario continuar la búsqueda de una mejor solución. Por esta razón, en el paso 3, si todas las tareas han sido asignadas, el algoritmo termina.

3. **Filtrar el conjunto de tareas previamente a la ejecución del algoritmo, para disminuir la cantidad de variables del problema, si fuera posible**. La mayoría de las veces, cuando hay una tarea que no puede ser asignada a ninguno de los procesadores, el algoritmo hace un gran número de iteraciones tratando de encontrar una asignación para todas las tareas. Basada en este hecho, la tercera modificación es filtrar el conjunto de tareas generadas y extraer aquellas que debido su tamaño, serán imposibles de asignar. Por ejemplo, si hay 4 procesadores y 10 tareas, 5 de las cuales tienen una utilización mayor a 0.5, será imposible asignar una de ellas, puesto que no se pueden dividir, y dos tareas con utilizaciones mayores a 0.5 no caben en un solo procesador debido al Teorema 2.10. En este ejemplo, el tamaño del problema sería de 40 variables ($40 = 4 \times 10$), lo cual define un espacio de búsqueda de $2^{40}$. Si se extrae una tarea, el tamaño del problema se reduce a 36 ($36 = 4 \times 9$), lo cual define un espacio de búsqueda de $2^{36}$. Este paso debe hacerse antes de todos los pasos de la Figura 4.3 y pudiera provocar una reducción exponencial en el espacio de búsqueda. Aunque el modelo de tareas con el cual se está trabajando es de plazos críticos, aunque se extraigan tareas con esta modificación, se ejecuta el algoritmo con el objetivo de no afectar las estadísticas de los resultados experimentales.

El algoritmo detallado con las modificaciones añadidas es el que se muestra en la Figura 4.5 en la página siguiente.

## 4.5. Ejemplo

A continuación, se expondrá un ejemplo simple de ejecución del método, paso a paso, con el objetivo de esclarecer su funcionamiento.

Supóngase que se tiene el siguiente conjunto de tareas $\tau = \{\tau_1, \ldots, \tau_8\}$ para ser asignadas a 2 procesadores. El conjunto de utilizaciones de las tareas es el siguiente, ordenado de forma decreciente:

$$\{0.35, 0.30, 0.30, 0.19, 0.18, 0.18, 0.15, 0.15\}. \tag{4.21}$$

La utilización total del conjunto es $U(\tau) = 1.8$. Nuestro problema tiene $8 \cdot 2 = 16$ variables binarias y $8 + 2 = 10$ restricciones. Las variables binarias se corresponden a las tareas como se muestra debajo:

$$\underbrace{X_1, X_2}_{\tau_1}, \underbrace{X_3, X_4}_{\tau_2}, \underbrace{X_5, X_6}_{\tau_3}, \underbrace{X_7, X_8}_{\tau_4}, \underbrace{X_9, X_{10}}_{\tau_5}, \underbrace{X_{11}, X_{12}}_{\tau_6}, \underbrace{X_{13}, X_{14}}_{\tau_7}, \underbrace{X_{15}, X_{16}}_{\tau_8}. \tag{4.22}$$

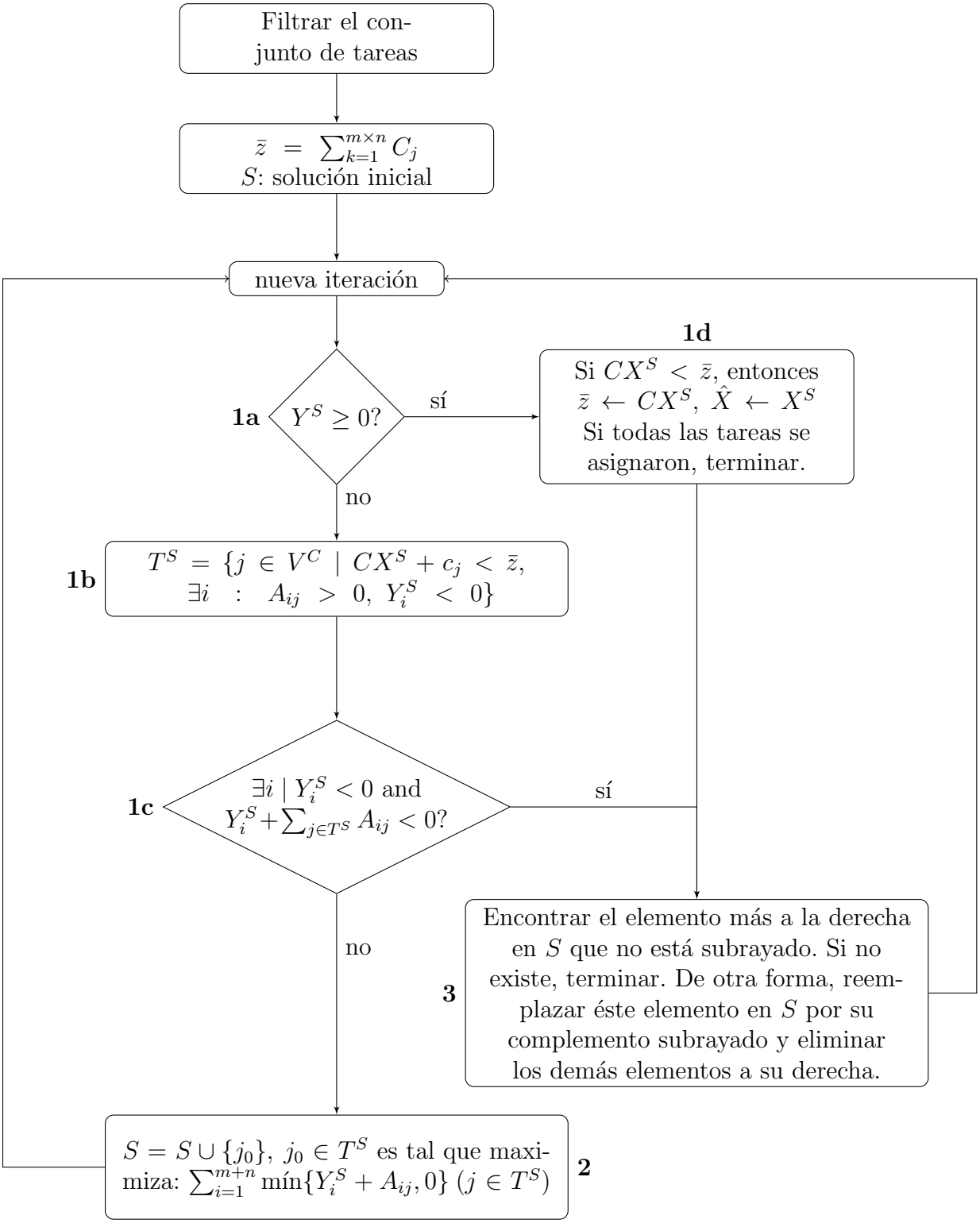


Figura 4.5: Algoritmo de Geoffrion aplicado al problema de la asignación.

Primero se procede a representar en un arreglo los coeficientes de la función objetivo y de la matriz $A$ y el vector $B$ de las restricciones (aquí ya se ha asumido el cambio de variables necesario para que el problema esté en la forma estándar requerida para aplicar el algoritmo):

$$C = [0.35, 0.35, 0.30, 0.30, 0.30, 0.30, 0.19, 0.19, 0.18, 0.18, 0.18, 0.18, 0.15, 0.15, 0.15, 0.15] \tag{4.23}$$

$$A = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0.35 & 0 & 0.3 & 0 & 0.3 & 0 & 0.19 & 0 & 0.18 & 0 & 0.18 & 0 & 0.15 & 0 & 0.15 & 0 \\ 0 & 0.35 & 0 & 0.3 & 0 & 0.3 & 0 & 0.19 & 0 & 0.18 & 0 & 0.18 & 0 & 0.15 & 0 & 0.15 \end{bmatrix} \tag{4.24}$$

$$B = [-1, -1, -1, -1, -1, -1, -1, -1, -0.8, -0.8]^T \tag{4.25}$$

La solución inicial se forma asignando la primera tarea al primer procesador, y la segunda al segundo procesador, con lo que la $S_0$ es:

$$S_0 = \{-1, 2, 3, -4\}. \tag{4.26}$$

Nótese que las variables $X_1$ y $X_2$ corresponden a $X_{1,1}$ y $X_{1,2}$, respectivamente. Originalmente, si se asigna la primera tarea al primer procesador, $X_1$ toma el valor de uno y $X_2$ de cero, pero como ya hemos asumido el cambio de variables, estos valores se invierten, quedando $X_1 = 0$ (representado por $-1$) y $X_2 = 1$ (representado por 2).

Las variables libres pertenecen al conjunto denotado por FreeV:

$$FreeV = \{5, 6, 7, \ldots, 15, 16\}. \tag{4.27}$$

La cota inicial $\bar{z}$ de la función objetivo es 3.6, que es el mayor valor que puede tomar $CX$. Esto sucede si todas las variables toman el valor de uno.

A continuación comienzan las iteraciones del algoritmo de la Fig. 4.5:

**iteración 1:**

$Y^S = [0, 0, -1, -1, -1, -1, -1, -1, -0.5, -0.45]^T$

$Y^S \ngeq 0$, entonces se va al paso de sondeo y se halla el conjunto $T^S$ de las variables que pudieran eliminar las infactibilidades de $Y^S$. Los índices donde $Y^S$ resultó negativo son: $2, 3, 4, 5, 6, 7, 8$ y 9. En este paso se busca en las filas de la matriz A (correspondientes a los elementos negativos de $Y^S$), si existe un índice de columna $j$ tal que $A_{ij} > 0$ y tal que $CX^S + c_j$ sea menor que la cota $\bar{z}$. La función objetivo evaluada en

la solución parcial $S$ es $CX^S = 0.65$. A continuación, se representan los coeficientes de $A$ correspondientes a las filas 3-10 y las columnas 5-16:

$$\left[\begin{array}{cccccccccccc} 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 & 16 \\ \hline \boxed{1} & \boxed{1} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \boxed{1} & \boxed{1} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \boxed{1} & \boxed{1} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \boxed{1} & \boxed{1} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \boxed{1} & \boxed{1} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \boxed{1} & \boxed{1} \\ 0.3 & 0 & 0.19 & 0 & 0.18 & 0 & 0.18 & 0 & 0.15 & 0 & 0.15 & 0 \\ 0 & 0.3 & 0 & 0.19 & 0 & 0.18 & 0 & 0.18 & 0 & 0.15 & 0 & 0.15 \end{array}\right] \tag{4.28}$$

Para todas las variables libres se cumple que $CX^S + c_j < \bar{z}$ y además, como se muestra arriba, para cada fila $i$ existe una columna $j$ (variable libre) tal que $A_{ij} > 0$. Por eso, en este caso, $T^S$ coincide con el conjunto de variables libres. $T^S = \{5, 6, \ldots, 16\}$.

Por razones de espacio, sólo se darán los detalles del sondeo en esta iteración. En el paso 1c, todavía se verifica que no haya que hacer vuelta atrás, examinando para los índices donde $Y^S < 0$ si $Y_i^S + \sum_{j \in T^S} A_{ij} < 0$.

índice 2: (tercera fila) $-1 + 2 = 1 \not< 0$

índice 3: (cuarta fila) $-1 + 2 = 1 \not< 0$

índice 4: (quinta fila) $-1 + 2 = 1 \not< 0$

índice 5: (sexta fila) $-1 + 2 = 1 \not< 0$

índice 6: (séptima fila) $-1 + 2 = 1 \not< 0$

índice 7: (octava fila) $-1 + 2 = 1 \not< 0$

índice 8: (novena fila) $-0.5 + 0.3 + 0.19 + 0.18 + 0.18 + 0.15 + 0.15 = 0.65 \not< 0$

índice 9: (décima fila) $-0.45 + 0.3 + 0.19 + 0.18 + 0.18 + 0.15 + 0.15 = 0.7 \not< 0$

Como ninguno de estos valores fue negativo, significa que no existe ningún índice en $Y^S$ para el cual sea imposible eliminar la infactibilidad. Luego se va al paso 2 para añadir una variable libre a $S$.

El criterio para añadir una variable a $S$ es que sea el $j \in T^S$ que maximice la expresión: $\sum_{i=1}^{10} \text{mín}\{Y_i^S + A_{ij}, 0\}$. Calculemos estos valores para todos los valores de $j$:

$j = 4$: (columna 5) $\text{mín}\{-1 + 1, 0\} + \text{mín}\{-1 + 0, 0\} + \text{mín}\{-1 + 0, 0\} + \text{mín}\{-1 + 0, 0\} + \text{mín}\{-1 + 0, 0\} + \text{mín}\{-1 + 0, 0\} + \text{mín}\{-0.5 + 0.3, 0\} + \text{mín}\{-0.45 + 0, 0\} = 0 - 1 - 1 - 1 - 1 - 1 - 0.2 - 0.45 =$ $\boxed{\text{-5.65}}$

$j = 5$: (columna 6) $\text{mín}\{-1 + 1, 0\} + \text{mín}\{-1 + 0, 0\} + \text{mín}\{-1 + 0, 0\} + \text{mín}\{-1 +$

$0,0\} + \text{mín}\{-1+0,0\} + \text{mín}\{-1+0,0\} + \text{mín}\{-0.5+0,0\} + \text{mín}\{-0.45+0.3,0\} = 0-1-1-1-1-1-0.5-0.15 = -5.65$

$j = 6$: (columna 7) $\text{mín}\{-1+0,0\} + \text{mín}\{-1+1,0\} + \text{mín}\{-1+0,0\} + \text{mín}\{-1+0,0\} + \text{mín}\{-1+0,0\} + \text{mín}\{-1+0,0\} + \text{mín}\{-0.5+0.19,0\} + \text{mín}\{-0.45+0,0\} = -1+0-1-1-1-1-0.31-0.45 = -5.76$

$j = 7$: (columna 8) $\text{mín}\{-1+0,0\} + \text{mín}\{-1+1,0\} + \text{mín}\{-1+0,0\} + \text{mín}\{-1+0,0\} + \text{mín}\{-1+0,0\} + \text{mín}\{-1+0,0\} + \text{mín}\{-0.5+0,0\} + \text{mín}\{-0.45+0.19,0\} = -1+0-1-1-1-1-0.5-0.26 = -5.76$

$j = 8$: (columna 9) $\text{mín}\{-1+0,0\} + \text{mín}\{-1+0,0\} + \text{mín}\{-1+1,0\} + \text{mín}\{-1+0,0\} + \text{mín}\{-1+0,0\} + \text{mín}\{-1+0,0\} + \text{mín}\{-0.5+0.18,0\} + \text{mín}\{-0.45+0,0\} = -1-1+0-1-1-1-0.32-0.45 = -5.77$

$j = 9$: (columna 10) $\text{mín}\{-1+0,0\} + \text{mín}\{-1+0,0\} + \text{mín}\{-1+1,0\} + \text{mín}\{-1+0,0\} + \text{mín}\{-1+0,0\} + \text{mín}\{-1+0,0\} + \text{mín}\{-0.5+0,0\} + \text{mín}\{-0.45+0.18,0\} = -1-1+0-1-1-1-0.5-0.27 = -5.77$

$j = 10$: (columna 11) $\text{mín}\{-1+0,0\} + \text{mín}\{-1+0,0\} + \text{mín}\{-1+0,0\} + \text{mín}\{-1+1,0\} + \text{mín}\{-1+0,0\} + \text{mín}\{-1+0,0\} + \text{mín}\{-0.5+0.18,0\} + \text{mín}\{-0.45+0,0\} = -1-1-1+0-1-1-0.32-0.45 = -5.77$

$j = 11$: (columna 12) $\text{mín}\{-1+0,0\} + \text{mín}\{-1+0,0\} + \text{mín}\{-1+0,0\} + \text{mín}\{-1+1,0\} + \text{mín}\{-1+0,0\} + \text{mín}\{-1+0,0\} + \text{mín}\{-0.5+0,0\} + \text{mín}\{-0.45+0.18,0\} = -1-1-1+0-1-1-0.5-0.27 = -5.77$

$j = 12$: (columna 13) $\text{mín}\{-1+0,0\} + \text{mín}\{-1+0,0\} + \text{mín}\{-1+0,0\} + \text{mín}\{-1+0,0\} + \text{mín}\{-1+1,0\} + \text{mín}\{-1+0,0\} + \text{mín}\{-0.5+0.15,0\} + \text{mín}\{-0.45+0,0\} = -1-1-1-1+0-1-0.35-0.45 = -5.8$

$j = 13$: (columna 14) $\text{mín}\{-1+0,0\} + \text{mín}\{-1+0,0\} + \text{mín}\{-1+0,0\} + \text{mín}\{-1+0,0\} + \text{mín}\{-1+1,0\} + \text{mín}\{-1+0,0\} + \text{mín}\{-0.5+0,0\} + \text{mín}\{-0.45+0.15,0\} = -1-1-1-1+0-1-0.5-0.3 = -5.8$

$j = 14$: (columna 15) $\text{mín}\{-1+0,0\} + \text{mín}\{-1+0,0\} + \text{mín}\{-1+0,0\} + \text{mín}\{-1+0,0\} + \text{mín}\{-1+0,0\} + \text{mín}\{-1+1,0\} + \text{mín}\{-0.5+0.15,0\} + \text{mín}\{-0.45+0,0\} = -1-1-1-1-1+0-0.35-0.45 = -5.8$

$j = 15$: (columna 16) $\text{mín}\{-1+0,0\} + \text{mín}\{-1+0,0\} + \text{mín}\{-1+0,0\} + \text{mín}\{-1+0,0\} + \text{mín}\{-1+0,0\} + \text{mín}\{-1+1,0\} + \text{mín}\{-0.5+0,0\} + \text{mín}\{-0.45+0.15,0\} = -1-1-1-1-1+0-0.5-0.3 = -5.8$

Se le añade la variable $X_5$ con valor de uno a $S$, por lo que ahora $S = \{-1, 2, 3, -4, 5\}$.

**iteración 2:**

$Y^S = [0, 0, 0, -1, -1, -1, -1, -1, -0.2, -0.45]^T$. Obsérvese que hay un elemento en

$Y^S$ que dejó de ser negativo (tercera posición) y otro que es ahora menos negativo (penúltima posición).

$Y^S \not\geq 0$, entonces se va al paso de sondeo. $T^S = \{7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 6\}$.

Para los índices en que $Y^S$ es negativo, se halla la suma del paso 1c. Como ninguna resulta negativa, se procede al paso 2 para añadirle una variable libre a $S$.

La variable que se le añade a $S$ es 7, con lo que $S = \{-1, 2, 3, -4, 5, 7\}$.

**iteración 3:**

$Y^S = [0, 0, 0, 0, -1, -1, -1, -1, -0.01, -0.45]^T$. Cada vez el vector $Y^S$ está más cerca de ser no negativo.

Como $Y^S \not\geq 0$, se procede a los pasos 1b y 1c. $T^S = \{9, 10, 11, 12, 13, 14, 15, 16, 6, 8\}$. Como ninguna suma del paso 1c resulta negativa para ningún índice en que $Y^S$ fue negativo, se procede al paso 2.

La variable que se añade a $S$ en el paso 2 es 10 (con valor de uno), por lo que ahora $S = \{-1, 2, 3, -4, 5, 7, 10\}$.

**iteración 4:**

$Y^S = [0, 0, 0, 0, 0, -1, -1, -1, -0.01, -0.27]^T$.

$Y^S \not\geq 0$, entonces se procede al sondeo. $T^S = \{11, 12, 13, 14, 15, 16, 9, 6, 8\}$. De igual forma que en las iteraciones anteriores, ninguna suma del paso 1c es negativa para los índices donde $Y^S$ fue negativo, con lo que se procede al paso 2.

La variable que se le añade ahora a $S$ es 12 con el valor de uno.
$S = \{-1, 2, 3, -4, 5, 7, 10, 12\}$.

**iteración 5:**

$Y^S = [0, 0, 0, 0, 0, 0, -1, -1, -0.01, -0.09]^T$.

$Y^S \not\geq 0$, entonces se procede al sondeo. $T^S = \{13, 14, 15, 16, 9, 11, 6, 8\}$. Como ninguna suma del paso 1c resulta negativa para ningún índice en que $Y^S$ fue negativo, se procede al paso 2.

La variable que se le añade ahora a $S$ es 14 con el valor de uno.
$S = \{-1, 2, 3, -4, 5, 7, 10, 12, 14\}$.

**iteración 6:**

$Y^S = [0, 0, 0, 0, 0, 0, 0, -1, -0.01, 0.06]^T$. Solamente quedan dos índices en $Y^S$ con elementos negativos.

Como aún $Y^S \not\geq 0$, se procede al sondeo. $T^S = \{15, 16, 9, 11, 13\}$. Como ninguna suma

del paso 1c resulta negativa para ningún índice en que $Y^S$ fue negativo, se procede al paso 2.

La variable que se le añade ahora a $S$ es 15 con el valor de uno.
$S = \{-1, 2, 3, -4, 5, 7, 10, 12, 14, 15\}$.

**iteración 7:**

$Y^S = [0, 0, 0, 0, 0, 0, 0, 0, 0.14, 0.06]^T$. Ahora $Y^S$ es no negativo, y se va al paso 1d, puesto que se ha encontrado una solución factible. Como el valor de la función objetivo (1.8) en esta solución parcial es menor que la cota $\bar{z}$, se actualiza la cota, y se almacena la solución $S$ como la mejor solución factible hasta el momento. Luego se verifica si todas las tareas han sido asignadas (se considera que las variables libres, por defecto, toman el valor de cero). Como éste es el caso, el algoritmo termina y su salida es:

```
0       1
1       0
1       0
1       0
0       1
0       1
0       1
1       0
se planificaron 8 de 8 tareas
```

Las filas del resultado se corresponden con la cantidad de tareas, y las columnas con la cantidad de procesadores. La última fila, por ejemplo, indica que $\tau_8$ ha sido asignada al procesador 2 y no al 1 (recuérdese que los valores de las variables están invertidos). En resumen, la asignación se ha hecho de la siguiente forma:

Al procesador 1: $\tau_1, \tau_5, \tau_6, \tau_7$, con utilizaciones correspondientes de $0.35, 0.18, 0.18$ y $0.15$.

Al procesador 2: $\tau_2, \tau_3, \tau_4, \tau_8$, con utilizaciones correspondientes de $0.30, 0.30, 0.19$ y $0.15$.

## 4.6. Detalles de implementación

A continuación se copian algunas funciones importantes de la implementación de la versión de Geoffrion adaptada para el problema de la asignación. Las variables que se utilizan son:

```
#define M 4                     // numero de procesadores disponibles
#define bound 80                // cota para cantidad de tareas
int cont = 0;                   // numero de tareas generadas
double Utotal;                  // utilizacion total del sistema
```

```
Ut[bound];                      // arreglo de utilizaciones del sist.
unsigned char v[M*bound];       // arreglo de variables binarias
                                // para mejor solucion almacenada
double B[bound + M];            // vector B de las restricciones
double A[bound + M][bound*M];   // matriz de restricciones
double YS[bound + M];           // vector Y^S = B + AX
double C[bound*M];              // vector de la funcion objetivo
int numPartial;                 // cantidad de elems en la
                                // mejor solucion almacenada
int TS[bound*M];                // arreglo para el conjunto T^S
int numTS;                      // cantidad de elementos en T^S
int freeV[bound*M];             // arreglo para las variables libres
int numFree;                    // cantidad de variables libres
double Ufiltered[bound-M];      // conjunto de tareas extraidas
int numFiltered;                // cantidad de tareas extraidas
FILE * Res;                     // archivo para escribir resultados
%int testNumber = 0;
```

Función donde se ejecuta el algoritmo de Geoffrion sobre un conjunto de tareas aleatoriamente generado:

```
void geoffrion(void){
  int i, n, s, j, mul;
  int realCont;
  uint64 p;             // unsigned int de 64 bits para el
                        // numero de iteraciones
  unsigned int pHi, pLo;
  int rows;             // cantidad de filas de la matriz
  int numS;             // cantidad de variables en la solucion
                        // parcial S
  int BinVar;           // cantidad de variables binarias
  int NR;               //
  int index;            // para el primer indice de elemento
                        // negativo en Y^S
  int el;               // para localizar primer elemento
                        // no subrayado en el paso 3
  double z;             // para cota de funcion objetivo
  int S[bound*M];       // arreglo para la solucion parcial
  unsigned char underS[bound*M+1];// arreglo para subrayados de S
  double d, por;
  int flag;             // para una bandera
  uint64 TS1, TS2;      // para leer tiempo inicial y tiempo final
  double execTime;
  double minim = 0.1;  // minima utilizacion posible para
```

```
                      // el conjunto de tareas
 generateTasks();     // genera tareas y forma Ut
 quicksort(Ut, 0, cont - 1);    // ordena Ut de forma decreciente
 if (Ut[cont - 1] < minim ) {   // si ultima menor que minima
    Ut[cont-2] += Ut[cont - 1]; // ultima se anade a penultima
    cont = cont - 1;            // hay una menos
  }
 quicksort(Ut, 0, cont - 1);
 filterUt();
// se visualiza el conjunto de utilizaciones generadas
 for(i = 0; i < cont; i ++){
   printf("%f \n", Ut[i]);
 }
 // se visualizan utilizaciones de las tareas extraidas por filtrado
 printf("extraidas\n:");
 for(i = 0; i < numFiltered; i++){
   printf("%f \n", Ufiltered[i]);
 }
 printf("\n");
 BinVar = M*cont;                  // cantidad de variables binarias
 initialSolution(BinVar, underS); // se forma la solucion
                                   // inicial (vector v)
 createB(rows);                    // se forma el vector B
 index = -1;                       // indice imposible
 createA(rows, BinVar);            // se forma la matriz A
 createC(BinVar);                  // se forma el vector objetivo C
 numS = initialS(S, M*M);          // se forma S
 z = M*Utotal;                     // cota inicial de funcion objetivo
 numFree = freeVar(S, numS, BinVar); // forma conjunto de var. libres
 #ifdef balas
 p = 1;                           // contador para iteraciones
 TS1 = cycles();                   // para medir tiempo inicial
 while (1){
   //Paso 1a
   CalculateYS(S, numS, rows);    // se calcula el vector Y^S
   if (testPositivenessYS(YS, rows, &index) == 0){
     //Paso 1b (Y^S NO es mayor o igual a cero)
     i = 0;                        // para moverse en filas de la matriz
     if (numFree != 0){     // si hay variables libres, se
                                   // halla el conjunto T^S
       numTS = findTS(z, S, numS, index, BinVar, rows);
     }
     else numTS = 0;
```

```
    // Paso 1c
    flag = 0;
    i = index;                  // primer indice negativo en Y^S
    // mientras no haya suma negativa y no terminen las filas
    while((flag == 0)&&(i < rows)){
      if (YS[i] < -0.000001) { // para los indices dde Y^S<0
        d = testSum(i);         // halla suma de paso 1c
        if(d < -0.00001) {      // si es negativa,
          flag = 1;             // levanta bandera
        }
      }
      i ++;                     // pasa a otra fila
    } // fin del while
    if (flag == 0) {            // si bandera no se levanto,
      //Paso 2: anadir variable a S
      augmentS(S, &numS, index, rows, underS);
      p++;                      // nueva iteracion
      continue;
    }
  }
  else{ // (Y^S fue mayor o igual a cero)
    //Paso 1d
    if (testForNewBound(S, numS, &z)){ // nueva cota?
      actualizeSolution(S, numS, underS); // actualiza solucion
      fillSolution(BinVar);             // llena solucion binaria
      // si todas se asignaron, terminar
      if (allAreScheduled(BinVar)) break;
    }
  } // fin del else
  // Paso 3
  el = locateElement(underS, numS);
  if (el == -1) {               // no existe, terminar
    break;
  }
  replaceAndDrop(S, &numS, underS, el);
  p ++;                         // nueva iteracion
}
TS2 = cycles();                 // para medir tiempo final
TS2 -= TS1;
execTime = (double)TS2/CPU_FREC; // calcula tiempo en segundos
fillSolution(BinVar);           // forma la solucion final
n = 0; mul = 1;
for(i = 0; i < BinVar; i ++){
```

```
    printf("%i\t",v[i]);
    if (i == mul*M-1) { printf("\n"); mul++;}
  }
  for(i = 0; i < cont; i ++){    // por tarea  (bloque)
    s = 0;
    for(j = 0; j < M; j++){
      if (v[i*M+j] == 1) s++;
    }
    if (s == M-1) n++;           // esa tarea fue planificada
  }
  realCont= cont + numFiltered;
  printf("se planificaron %i de %i tareas", n, realCont);
  por = (double)(n*100)/realCont;
  pLo = (unsigned int)p;
  pHi = (unsigned int)(p>>32);
  fprintf(Res, "%d\t", realCont);
  fprintf(Res, "%f\t", por);
  fprintf(Res, "%f\t" , execTime);
  fprintf(Res, "%u\t%u\n", pHi, pLo);
  #endif
}
```

Funciones para crear el vector de la función objetivo $C$ y la matriz $A$ y el vector $B$ de las restricciones:

```
// crea matriz de restricciones de dimension (cont+M)x(BinVar).
// O sea, tiene BinVar columnas y (cont + M) filas.
void createA(int rows, int BinVar){
  int i, j;
  // como es muy esparcida primero se llena todo de ceros
  for(i = 0; i < rows; i ++){
    for(j = 0; j < BinVar; j ++){
      A[i][j] = 0;
    }
  }
  // las primeras cont filas contienen m unos dependiendo del
  // bloque de tamano m en que este
  for(i = 0; i < cont; i ++){
    for(j = i*M; j <= (i+1)*M - 1; j ++){
      A[i][j] = 1;
    }
  }
  // las restantes M filas son bloques de tamano MxM
  // que son matrices diagonales con la utilizacion de la
```

```
  // tarea correspondiente al bloque (1er bloque-1ra tarea)
  for(i = cont; i < rows; i ++){
    for(j = 0; j < cont; j ++){
      A[i][j*M + i - cont] = Ut[j];
    }
  }
}

// Crea el vector B de terminos independientes de las restricciones.
// Tiene dimension (cont+M)x1.
void createB(int rows){
  int i;
  // los primeros cont elementos tienen el valor de (1-M)
  for(i = 0; i < cont; i ++){
    B[i] = 1 - M;
  }
  // los restantes M elementos tienen el valor de (1-Utotal)
  for(i = cont; i < rows; i ++){
    B[i] = 1 - Utotal;
  }
}

// Crea el vector de coeficientes de la funcion objetivo.
// Como hay (cont*M) variables, los primeros M coeficientes tienen
// el valor de U1 pues corresponden a la tarea 1, los segundos M
// coeficientes tienen el valor de U2 y asi sucesivamente.
void createC(int BinVar){
  int i;
  for(i = 0; i < BinVar; i ++){
    C[i] = Ut[(int)(i/M)];
  }
}
```

La siguiente función forma el vector $S$ de la solución inicial como se describió en las modificaciones al algoritmo:

```
// Halla solucion inicial. Una solucion inicial factible es
// asignar la primera tarea al procesador 1, la 2da al 2do y
// asi, asignar las primeras M tareas de mayor utilizacion.
// Pero como se ha hecho un cambio de variables para adaptar
// el problema, ahora se trabajara con los complementos.
void initialSolution(int n, unsigned char underS[]){
  int i;
  // primero se llenan todas con unos
```

```
  for(i = 0; i < n; i ++){
    v[i] = 1;  // v es un arreglo que contiene los valores
               // asignados a las variables binarias
  }
  // la primera variable del primer bloque de tamano M,
  // la segunda del segundo bloque, etc, son cero.
  for(i = 0; i < M; i ++){
    v[i*M + i] = 0;
  }
  // Se llena el arreglo correspondiente a subrayados. Nada ha
  // sido subrayado aun.
  for(i = 0; i < n; i ++){
    underS[i] = 0;
  }
}
```

La función más importante del paso 1b (hallar el conjunto $T^S$):

```
// Se debe buscar en la fila index de la matriz A los indices de
// columnas j tal que Aij > 0 y tal que CXˆS + Cj < b (la cota),
// y almacenarlos en el arreglo TS. La funcion devuelve la can-
// tidad de elementos en TS.
int findTS(double b, int S[], int numS, int index, int BinVar, int rows){
  int j, c, k;
  c = 0;                          // contador de elementos en TS
  for(k = index; k < rows; k ++){ // se recorre todo YˆS, a par-
                                  // tir del 1er indice negativo
    if (YS[k] < -0.00001){        // si YˆS menor que cero
      for(j = 0; j < BinVar; j ++){
        // Si Aij > 0 y la variable es libre
        if ((A[k][j] > 0.00001)&&(!(notInS(j + 1, freeV, numFree)))){
          if (COnX(S, numS) + Ut[(int)(j/M)] < b){ // si CXˆS + Cj < b
            //aumenta TS
            if (!isInTS(j+1, TS, c)){  // si no esta ya en TS,
              TS[c] = j+1;             // se anade a TS
              c ++;
            }
          }
        }
      }// fin de segundo ciclo for
    }
  }// fin de primer ciclo for
  return c;
}
```

La función más importante del paso 2:

```
// Esta funcion anade a S la variable libre perteneciente al conjunto TS,
// que maximice la expresion \sum(min{0, YS_{i} + Aij}):
void augmentS(int S[], int *numS, int index, int rows, unsigned char underS[]){
  int i, j;
  double toStore, s;
  i = 1;
  // calcula suma para el primer elem. en TS y la almacena en toStore
  toStore = calculateSum(TS[0], rows);
  j = 0;
  while(i < numTS){                // mientras haya elementos en TS
    s = calculateSum(TS[i], rows); // calcula la suma
    if (s-0.000000001 > toStore) { // si es mayor que toStore
      toStore = s;                 // la almacena
      j = i;                       // se guarda el indice
    }
    i ++;
  }
  // en j esta el indice que maximizo la suma
  S[*numS] = TS[j];                // se anade variable a S
  *numS = *numS + 1;               // se aumenta numS
  underS[*numS] = 0;               // sin subrayado
  // se debe quitar esa variable del conjunto de libres
  releaseFree(TS[j]);
}
```

La función más importante del paso 3:

```
// Reemplaza el elemento que corresponde al indice index en S por su
// complemento y lo subraya. Luego, elimina todos los demas elementos
// de S que esten a su derecha. Estos se anaden a las var. libres.
void replaceAndDrop(int S[], int *numS, unsigned char underS[], int index){
  int i;
  underS[index] = 1;         // se subraya
  S[index] = (-1)*S[index];  // se reemplaza por su complemento
  for(i = index+1; i < *numS; i ++){
    // de ese elemento en adelante, se anaden a las libres
    if (S[i] > 0) addFree(S[i]); // siempre con signo positivo
    else addFree((-1)*S[i]);     // variable es ahora libre
  }
  *numS = index+1;             // se cambia tamano de S y underS
  underS[*numS] = 0;

}
```

Las funciones más importantes del paso 1d, aunque sencillas:

```
// Prueba si la solucion almacenada en S tiene un valor de la
// funcion objetivo menor que el incumbente. El parametro z es
// la cota actual de f. Devuelve 1 si se actualiza la cota y
// cero en caso contrario.
int testForNewBound(int S[], int numS, double *z){
  double b = 0;
  b = COnX(S, numS);             // se calcula f(X^S)
  if (b < *z) {                  // si es menor que cota
    *z = b;                      // actualiza cota
    return 1;
  } else return 0;
}

// Esta funcion devuelve 1 si todas las tareas fueron asignadas y
// cero sino. v es un arreglo de cont bloques de tamano M (cont =
// cantidad de tareas). En cada bloque, debe haber solo un cero
// para que todas las tareas hayan sido asignadas.
int allAreScheduled(int BinVar){
  int i, j, n, s;
  n = 0;
  for(i = 0; i < cont; i ++){   // por tarea  (bloque)
    s = 0;
    for(j = 0; j < M; j++){      // se suma la cantidad de 1s
      if (v[i*M+j] == 1) s++;
    }
    // si hay M-1 unos --> hay un solo cero en el bloque
    if (s == M-1) n++;           // esa tarea fue planificada
  }
  if (n == cont) return 1; else return 0;
}
```

La función de filtrado:

```
void filterUt() {
    int i;
    numFiltered=0;   // se analizan las M tareas de mayor utilizacion
    while( (M<cont) && (Ut[M]+Ut[M-1] > 1.000001)){
        // tarea M es imposible de planificar, quitarla
        Ufiltered[numFiltered++] = Ut[M]; // la anade a arreglo Ufiltered
        Utotal -= Ut[M];
        // recorrer tareas de Ut
        cont--;
        for(i=M; i<cont; i++){
```

```
            Ut[i] = Ut[i+1];
        }
    }
}
```

## 4.7. Restricciones de sustitución

Los criterios de sondeo expuestas en la página 57 de la sección 4.3.1 se aplican a las restricciones del problema, una a la vez. Para mitigar esta limitación, se pueden introducir periódicamente lo que se llaman restricciones de sustitución, como sugirió Glover [Glo65]. Estas restricciones son redundantes en el sentido que encapsulan toda la información contenida en las restricciones originales del problema. Sin embargo, los criterios de sondeo son efectivos al aplicarlas a ellas individualmente. En vez de aplicar los criterios de sondeo a un gran número de restricciones, sólo se aplican a unas pocas restricciones de sustitución, por lo cual se acelera el proceso de sondeo. Pero es necesario que los criterios de sondeo sean efectivos cuando se aplican a este tipo de restricciones, así que el que se usará es el siguiente [Geo69]:

- Criterio de infactiblidad binaria: la restricción $b + \sum_j a_j x_j > (\geq) 0$ no tiene solución binaria si, y sólo si:

$$b + \sum_j \text{máx}\{0, a_j\} \leq (<) 0. \tag{4.29}$$

**Definición 4.7.1.** *[Glo65]Sea el problema de programación lineal entera binaria:*

$$\text{mín } CX \tag{4.30}$$
$$s.a.\ AX \geq B, \tag{4.30a}$$
$$X \geq 0,\ x_i = 0\ \acute{o}\ 1, \tag{4.30b}$$

*donde* $X = (x_1, \ldots, x_n)$, $B = (b_1, \ldots, b_m)$, $A \in \mathbb{R}^{m \times n}$. *La restricción de sustitución se define como una combinación lineal no negativa de las restricciones 4.30a, en la cual a al menos una de las restricciones se le da un peso estrictamente positivo. Se representa por:*

$$aX \geq s, \tag{4.31}$$

*donde* $s$ *es un escalar y* $a$ *es un vector fila (*$s = uB$, $a = uA$, $u = (u_1, \ldots, u_m)^T$, *además* $u_i \geq 0\ \forall i$ *y* $u_i > 0$ *para al menos un índice* $i$*).*

En cada momento, se tiene un problema de programación entera asociado con la solución parcial $S$ correspondiente al nodo que se esté sondeando, al que denotaremos

(PS) de la siguiente forma[1]. Las variables de (PS) son las variables libres (en $V^C$).

$$\text{mín} \sum_{j\in S} c_j x_j^S + \sum_{j\in V^C} c_j x_j, \tag{4.32}$$

$$s.a.\ b_i + \sum_{j\in S} a_{ij} x_j^S + \sum_{j\in V^C} a_{ij} x_j \geq 0,\ \forall i \in [1,m], \tag{4.32a}$$

$$0 \leq x_j \leq 1 \text{ y enteras},\ \forall j \in V^C. \tag{4.32b}$$

En [Geo69], se propone añadir al problema anterior algunas restricciones de sustitución de la siguiente forma:

$$\mu(B + AX) + (\bar{z} - CX) > 0, \tag{4.33}$$

donde $\mu$ es un $m$-vector no negativo, puesto que:

**Lema 4.7.1.** *La ecuación 4.33 se cumple para cualquier solución factible de 4.30 que tenga un mejor valor de $CX$ que $\bar{z}$.*

**Demostración:** En efecto, sea $\hat{S}$ una solución factible de 4.30 que provee un menor valor de $CX$ que $\bar{z}$. Entonces:

$$B + AX^{\hat{S}} \geq 0,\ 0 \leq X_j^{\hat{S}} \leq 1 \text{ y entero } \forall j. \tag{4.34}$$

También, por hipótesis, se cumple que:

$$CX^{\hat{S}} < \bar{z} \Longrightarrow \bar{z} - CX^{\hat{S}} > 0. \tag{4.35}$$

Como $\mu \geq 0$ y $B + AX^{\hat{S}} \geq 0$, entonces $\mu(B + AX^{\hat{S}}) \geq 0$.

Luego, $\mu(B + AX^{\hat{S}}) + (\bar{z} - CX^{\hat{S}}) > 0$, que es lo que se quería demostrar. ■

Esto es parte de un lema [Glo65] de donde parte la definición de cuán "fuerte" es una restricción de sustitución. Considérese el problema:

$$\begin{aligned} &\text{mín } CX \\ &s.a.\ aX \geq s,\ \text{(restricción de sustitución)} \\ &\quad X \geq 0,\ x_i = 0 \text{ ó } 1. \end{aligned} \tag{4.36}$$

Un resultado del lema antes mencionado es que si $\bar{X}$ es solución óptima de 4.36 y $X^*$ es solución óptima de 4.30, entonces $C\bar{X} \leq CX^*$. Por lo que, dadas dos restricciones de sustitución para el problema 4.30, se dice que la primera es más fuerte que la segunda si la solución óptima de 4.36 obtenida con la primera restricción conlleva a un mayor valor de la función objetivo que la solución óptima de 4.36 obtenida con la segunda restricción (pues se obtendría una "cota"más ajustada para el valor óptimo

[1] $x_j^S$ es el valor de la variable $j$-ésima en la solución parcial $S$. Es decir, $x_j^S = 1$ (0) si $j(-j) \in S$.

de 4.30).

Ahora bien, debido a que en cualquier instante del cálculo, hay un conjunto de variables libres respecto a la solución parcial $S$, la "fuerza"de la restricción de sustitución se define con respecto a $S$. Para que el criterio de sondeo de infactibilidad binaria definido en la Ecuación 4.29 sea efectivo al aplicarlo a las restricciones de sustitución, la "fuerza"de éstas debe ser definida de acuerdo con cuán cerca esté la restricción de ser binaria infactible.

**Definición 4.7.2.** *[Geo69] La restricción de sustitución* $\mu^1(B+AX)+(\bar{z}-CX)>0$ *es más fuerte que la restricción* $\mu^2(B+AX)+(\bar{z}-CX)>0$ *(relativa a la solución parcial* $S$*) si el máximo del lado izquierdo de la primera restricción es menor que el máximo del lado izquierdo de la segunda, tomando el máximo sobre los valores binarios de las variables libres.*

Entonces, encontrar una restricción más fuerte es minimizar sobre todos los $\mu \geq 0$, la expresión:

$$\text{máx}\Big\{\mu(B+AX)+(\bar{z}-CX) \mid \ x_j = 0 \text{ ó } 1(j \notin S) \text{ y } x_j = x_j^S (j \in S)\Big\}. \tag{4.37}$$

Esto es equivalente a:

$$\sum_{i=1}^{m} \mu_i b_i^S + \bar{z} - z^S + \text{máx}\Big\{\sum_{j\notin S}\Big(\sum_{i=1}^{m}\mu_i A_{ij} - c_j\Big)x_j \ \mid \ x_j = 0,1(j \notin S)\Big\}, \tag{4.38}$$

donde:

$$z^S = \sum_{j\in S} c_j x_j^S, \tag{4.39}$$

$$b_i^S = b_i + \sum_{j\in S} A_{ij} x_j^S. \tag{4.40}$$

Para todo $\mu \geq 0$, se tiene:

$$\begin{aligned} &\text{máx}\Big\{\sum_{j\notin S}\Big(\sum_{i=1}^{m}\mu_i A_{ij} - c_j\Big)x_j \ \mid \ x_j = \{0,1\}(j \notin S)\Big\} \\ =&\text{máx}\Big\{\sum_{j\notin S}\Big(\sum_{i=1}^{m}\mu_i A_{ij} - c_j\Big)x_j \ \mid \ 0 \leq x_j \leq 1 \ (j \notin S)\Big\}, \end{aligned} \tag{4.41}$$

puesto que el máximo de una función lineal se va a alcanzar en alguno de los valores extremos de la $x_j$ (0 ó 1). Luego, se tiene el problema con $|V^C|$ variables y $|V^C|$ restricciones:

$$\text{máx}\sum_{j\notin S}\Big(\sum_{i=1}^{m}\mu_i A_{ij} - c_j\Big)x_j \tag{4.42}$$

$$s.a.\ x_j \leq 1 \ (j \notin S), \tag{4.42a}$$

$$x_j \geq 0. \tag{4.42b}$$

El correspondiente problema dual es el siguiente:

$$\text{mín} \sum_{j \notin S} \omega_j \tag{4.43}$$

$$s.a.\ \omega_j \geq \sum_{i=1}^{m} \mu_i A_{ij} - c_j \ \ (j \notin S), \tag{4.43a}$$

$$\omega_j \geq 0. \tag{4.43b}$$

Sustituyendo (4.43) en (4.38), tenemos que se trata de minimizar, sobre todos los $\mu$, la expresión:

$$\sum_{i=1}^{m} \mu_i b_i^S + \bar{z} - z^S + \text{mín} \Big\{ \sum_{j \notin S} \omega_j \ \mid \ \omega_j \geq 0 \text{ y } \omega_j \geq \sum_{i=1}^{m} \mu_i A_{ij} - c_j \ \ (j \notin S) \Big\}. \tag{4.44}$$

Denotemos por (LPs) el siguiente problema de optimización:

$$\min_{\mu,\omega} \sum_{i=1}^{m} \mu_i b_i^S + \sum_{j \notin S} \omega_j + \bar{z} - z^S \tag{4.45}$$

$$s.a.\ \omega_j \geq \sum_{i=1}^{m} \mu_i A_{ij} - c_j \ \ (j \notin S), \tag{4.45a}$$

$$\omega_j \geq 0 \ \ (j \notin S), \tag{4.45b}$$

$$\mu_i \geq 0 \ \ (i = 1, \ldots, m). \tag{4.45c}$$

La versión continua del problema original (PS) (4.32) es la siguiente:

$$\text{mín} \sum_{j \in S} c_j x_j^S + \sum_{j \notin S} c_j x_j \tag{4.46}$$

$$s.a.\ \sum_{j \in S} A_{ij} x_j^S + \sum_{j \notin S} A_{ij} x_j \geq -b_i \ \ (i = 1, \ldots, m), \tag{4.46a}$$

$$\left.\begin{array}{l} x_j \leq 1, \\ x_j \geq 0 \end{array}\right\} (j \notin S). \tag{4.46b}$$

Este último problema es equivalente a:

$$\text{mín} \sum_{j \notin S} c_j x_j^S \equiv \text{máx} - \sum_{j \notin S} c_j x_j^S \tag{4.47}$$

$$s.a.\ -\sum_{j \notin S} A_{ij} x_j \leq \underbrace{\sum_{j \in S} A_{ij} x_j^S + b_i}_{b_i^S} \ \ (i = 1, \ldots, m), \tag{4.47a}$$

$$\left.\begin{array}{l} x_j \leq 1, \\ x_j \geq 0 \end{array}\right\} (j \notin S). \tag{4.47b}$$

El dual de esta versión continua de (PS) es como sigue (con $m + |V^C|$ variables que se denotarán $\mu_1, \ldots, \mu_m$ y $\omega_j$ $(j \notin S)$):

$$\text{mín} \sum_{i=1}^{m} \mu_i b_i^S + \sum_{j \notin S} \omega_j \tag{4.48}$$

$$s.a. \ -\sum_{j \notin S} A_{ij}\mu_i + \omega_j \geq -c_j \ \ (j \notin S), \tag{4.48a}$$

$$\omega_j \geq 0 \ \ (j \notin S), \tag{4.48b}$$

$$\mu_i \geq 0 \ \ (i = 1, \ldots, m). \tag{4.48c}$$

De aquí, se ve que el dual de la versión continua de (PS) (4.48) es equivalente al problema de optimización (LPs) (4.45). Por lo tanto, las variables óptimas **duales** de (LPs) son óptimas en (PS) si son todas enteras.

**Proposición 4.7.1.** *[Geo69] Sea $S$ una solución parcial arbitraria del problema de optimización (P) (4.30). Entonces, (LPs) (4.45) es forzosamente factible, y se cumple:*

1. *El valor óptimo de (4.45) es menor o igual a cero si, y sólo si, existe una restricción de sustitución binaria infactible.*

2. *El valor óptimo de (4.45) es mayor que cero si, y sólo si, no existe ninguna restricción de sustitución que sea binaria infactible, y el vector óptimo $\mu$ conduce a una restricción de sustitución más fuerte, relativa a $S$.*

Por lo tanto, lo que se hace es ejecutar iteraciones de simplex u otro método de los descritos en el Capítulo 3 hasta que sucede una de las tres opciones siguientes, mutuamente excluyentes:

(a) El valor de la función objetivo de (LPs) se vuelve menor o igual a cero. Entonces, existe una restricción de sustitución que es binaria infactible y se puede podar la rama. Ir al paso 4. En [Geo69] añaden una restricción de sustitución de todas maneras en este caso.

(b) Se obtiene una solución óptima de (LPs), el valor óptimo de (LPs) es mayor que cero y las variables duales óptimas de (LPs) son enteras. La solución óptima de (PS) está dada por los valores de las variables duales, por lo que se puede reemplazar la cota y actualizar la mejor solución almacenada. Ir al paso 4.

(c) Se obtiene una solución óptima de (LPs), el valor óptimo de (LPs) es mayor que cero, pero no todas las variables duales óptimas de (LPs) son enteras. Se obtiene una restricción de sustitución "más fuerte çon el vector de variables óptimas $\mu$, la cual se añade al problema original.

Entonces, el algoritmo para resolver el problema (P) por enumeración implícita usando las restricciones de sustitución es el siguiente[2]:

[2]Este algoritmo termina en un número finito de pasos con una solución óptima de (P) ó concluye que ésta solución óptima no existe [Geo69].

**Paso 0:** Inicializar $\bar{z}$ en una cota superior conocida del valor óptimo de (P). Inicializar $S$ en una solución parcial arbitraria (sin subrayados).

**Paso 1:** Si (PS) tiene una solución óptima obvia con valor menor a $\bar{z}$, entonces, reemplazar $\bar{z}$ por ese valor, guardar la solución e ir al Paso 4.

Si (PS) no tiene ninguna solución factible que mejore la cota $\bar{z}$, ir al paso 4. Esto se hará examinando si la(s) restricción(es) de sustitución agregadas son binarias infactibles aplicando el criterio de sondeo 4.29. Si aún no se ha añadido ninguna, usar los criterios del algoritmo 4.5 en las restricciones originales.

Si cualquiera de las variables libres debe tomar el valor de 1 ó de 0 para que (PS) tenga una solución factible que provea un valor menor que $\bar{z}$, entonces aumentar $S$ por $j$ ó $-j$ para cada variable que deba tomar el valor de 1(0). En cualquier solución binaria de $b + \sum_j a_j x_j > (\geq)0$:

$$b + \sum_j \text{máx}\{0, a_j\} - \mid a_{j_0} \mid \leq (<)0 \text{ implica que } x_{j_0} = 0(1) \text{ si } a_{j_0} < (>)0. \quad (4.49)$$

**Paso 2:** Resolver (LPs). Si el óptimo de (LPs) se vuelve no positivo, ir al paso 4. En caso contrario, analizar las variables duales óptimas de (LPs). Si son todas enteras, reemplazar la cota, actualizar la mejor solución almacenada, e ir al paso 4. Sino, se pueden redondear los valores de las variables duales a los valores enteros más cercanos, en un intento de acertar en la solución óptima, ó solamente agregar la restricción.

Eliminar una o más restricciones de sustitución, según se decida. En [Geo69] se trabaja con las cuatro últimas restricciones agregadas.

**Paso 3:** Aumentar $S$ por el valor de cero o uno de alguna variable libre. La variable que se escogerá, será, entre todas las variables libres, la que maximice la expresión:

$$\sum_{i=1}^{m} \text{mín}\{0, B_i^S + A_{ij}\}. \quad (4.50)$$

Regresar al **Paso 1**.

**Paso 4:** Localizar el elemento más a la derecha en $S$ que no esté subrayado. Si no existe, terminar. En caso contrario, reemplazarlo por su complemento subrayado y eliminar todos los elementos a su derecha. Regresar al **Paso 1**.

En nuestro problema de planificación multiprocesador, dada la estructura particular de la matriz de coeficientes $A_{ij}$, la restricción de sustitución $\mu(B+AX)+(\bar{z}-CX) > 0$

se calcula como se describe a continuación:

$$(\mu_1 \dots \mu_{m+n}) \left( \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_{m+n} \end{bmatrix} + \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1,m\times n} \\ a_{22} & a_{21} & \dots & a_{2,m\times n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m+n,1} & a_{m+n,2} & \dots & a_{m+n,m\times n} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_{m+n} \end{bmatrix} \right) + (\bar{z} - CX) > 0. \tag{4.51}$$

Lo cual es equivalente a:

$$\mu_1(b_1 + a_{11}x_1 + \dots + a_{1,m\times n}x_{m\times n}) + \mu_2(b_2 + a_{22}x_1 + \dots + a_{2,m\times n}x_{m\times n}) + \dots +$$
$$\mu_{m+n}(b_{m+n} + a_{m+n,1}x_1 + \dots + a_{m+n,m\times n}x_{m\times n}) + \bar{z} - \sum_{i=1}^{m\times n} c_i x_i > 0.$$

El término constante es:

$$T = \mu_1 b_1 + \mu_2 b_2 + \dots + \mu_{m+n} b_{m+n} + \bar{z}. \tag{4.52}$$

Nótese que en la matriz de coeficientes, en cada columna, que corresponde a una variable $x_i$, hay un uno y un valor correspondiente a una utilización del conjunto de utilizaciones de las tareas. Por esta razón, el coeficiente correspondiente al $x_i$ tiene la forma $\mu_k + \mu_p u_q - c_i$, donde $k, p$ y $q$ dependen de $i$. Los coeficientes correspondientes a a las variables $x_i$ se calculan como sigue:

```
k = 0;
desde i = 0 ... n hacer:
  desde j = 0 ... m hacer:
    CoefX[k] = mu[i] + U[i]*mu[n+j];
    k = k +1;
  fin de ciclo en j
fin de ciclo en i
CoefX[k] = CoefX[k] - c[k];
```

Para resolver (LPs), lo que se sugiere en [Geo69] es usar una subrutina de método Simplex Revisado con inverso explícito, incorporando técnicas que aprovechan los resultados de cálculos previos para los cálculos posteriores y así no tener que comenzar todo de nuevo cada que se ejecuta el paso 2. En estas técnicas utilizan un procedimiento de etiquetado en vez de hacer manipulaciones clásicas con matrices. No obstante, en problemas grandes sería más conveniente utilizar algún método de punto interior como el algoritmo de Mehrotra descrito en el Capítulo 3.

El algoritmo detallado modificado para incluir las restricciones de sustitución se muestra en Figura 4.6.

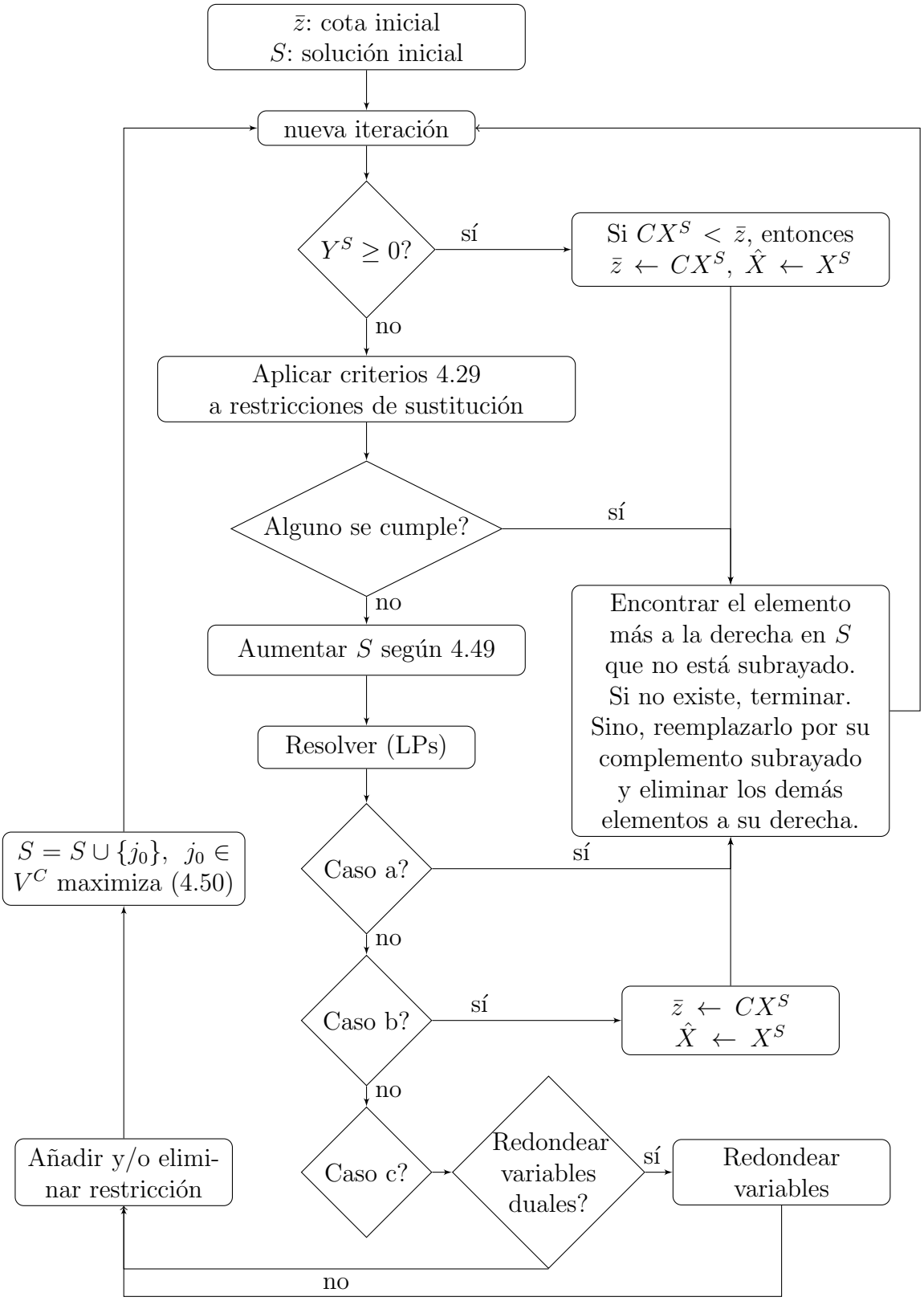


Figura 4.6: Algoritmo de Geoffrion con restricciones de sustitución.

# Capítulo 5

# Resultados Experimentales

Varios tipos de experimentos fueron llevados a cabo, los cuales se describen en las tres secciones de este Capítulo. Estos experimentos fueron realizados en su totalidad en una computadora personal Intel Core 2 Duo CPU, a 2.93 GHz de velocidad y con 3.0 GB de RAM. El sistema operativo de la computadora es Windows 7 de 32-bits. La aplicación fue programada en lenguaje C y compilada con GNU.

Se consideraron sistemas multiprocesadores con 4 y 6 procesadores, y se generaron varias muestras de conjuntos de tareas. Los períodos de las tareas se generaron tal que estuvieran uniformemente distribuidos en el intervalo [100, 500], y las utilizaciones de las tareas fueron generadas con distribución uniforme en los intervalos correspondientes a cada tipo de experimento. Además, se estudiaron los tiempos de respuesta del algoritmo para utilizaciones totales de los conjuntos de tareas variando desde el 50 % hasta el 100 % de la capacidad total del sistema multiprocesador. Esto es, para cada valor de $U_t \in \{2, 2.2, 2.4, \ldots, 3.8, 4\}$, se generaron aleatoriamente 500 conjuntos de tareas con el objetivo de evaluar el desempeño del algoritmo.

Para una muestra de conjuntos de tareas $TS$, definamos la *tasa de planificación* $\rho$ de un algoritmo de planificación multiprocesador $P$ como:

$$\rho_{TS} = \frac{\begin{array}{c}\text{cantidad de conjuntos de tareas} \\ \text{en que } P \text{ planificó todas las tareas}\end{array}}{\text{cantidad total de muestras en } TS} * 100. \tag{5.1}$$

O sea, se cuenta uno si todos las tareas fueron planificadas por el algoritmo en cada conjunto y cero en caso contrario. La tasa de planificación es el promedio de conjuntos de tareas donde todas las tareas fueron planificadas por $P$.

Para cada experimento, la tasa de planificación permite hacer conjeturas sobre las cotas de utilización para la factibilidad de la planificación de un conjunto de tareas bajo EDF usando esquema particionado, con las características del modelo de tareas propuesto.

En cada sección se muestra la gráfica de la tasa de planificación contra la utilización

total del sistema multiprocesador y la tabla de tiempos (en segundos) y número de iteraciones requeridas en promedio para obtener la asignación óptima.

No se pretende aquí comparar el desempeño del algoritmo propuesto con otros algoritmos. No sería comparable con un algoritmo heurístico, pues éste último no hallaría una planificación óptima, por lo que el algoritmo exacto sería "mejor"que el heurístico en este aspecto. Por otra parte, el algoritmo heurístico tendría menor tiempo de ejecución que un algoritmo exacto, por lo que resultan incomparables. Hasta donde conocemos, no existen resultados experimentales concernientes al tiempo de ejecución de algoritmos de planificación multiprocesador de tiempo real. El propósito de esta sección es ofrecer evidencias experimentales que demuestren la factibilidad de utilizar algoritmos de planificación multiprocesador que hallen una asignación óptima, para algunos sistemas prácticos.

## 5.1. 4 procesadores

### 5.1.1. Tareas con utilizaciones entre 0.1 y 0.7

En este experimento, se consideraron tareas con utilizaciones variando entre 0.1 y 0.7. Se generaron 500 conjuntos de tareas para cada valor de la utilización total del sistema multiprocesador, variando entre el 50 y el 100 %.

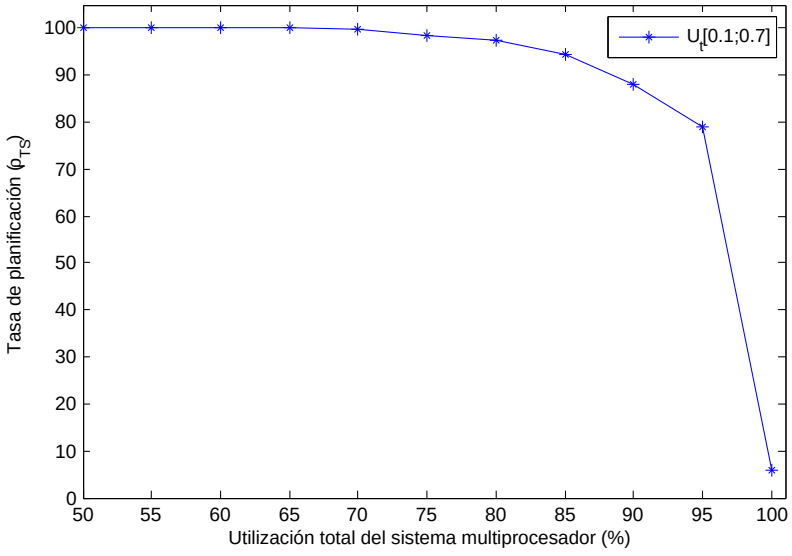


Figura 5.1: Tasa de planificación de los conjuntos de tareas (4 procesadores, $u_i \in [0.1, 0.7]$).

La Figura 5.1 permite conjeturar que quizás la cota de utilización planificable para que un conjunto de tareas de estas características sea planificable bajo EDF usando

esquema particionado en un sistema multiprocesador es del 65 % ó menor.

Para este experimento, donde se mezclan tareas que consumen poco tiempo de CPU y otras que consumen bastante tiempo de CPU, los tiempos son razonablemente bajos, considerando que el algoritmo toma en promedio 18 s en terminar para el peor caso[1], es decir, para el 100 % de la utilización total, como muestra la siguiente Tabla:

| $U_t$ (%) | # iterac. | $t$ (seg.) |
|---|---|---|
| 50 | 5.1 | $1.54 \cdot 10^{-5}$ |
| 55 | 6.5 | $2.3 \cdot 10^{-5}$ |
| 60 | 8 | $3.5 \cdot 10^{-5}$ |
| 65 | 9.5 | $6 \cdot 10^{-5}$ |
| 70 | 11 | $8 \cdot 10^{-5}$ |
| 75 | 57.6 | $4.3 \cdot 10^{-4}$ |
| 80 | 335 | $2.4 \cdot 10^{-3}$ |
| 85 | $1.6 \cdot 10^{3}$ | 0.012 |
| 90 | $3.9 \cdot 10^{3}$ | 0.03 |
| 95 | $1.5 \cdot 10^{4}$ | 0.125 |
| 100 | $1.6 \cdot 10^{6}$ | 17.8 |

Tabla 5.1: Tiempo (en segundos) y número de iteraciones (4 procesadores, $u_i \in [0.1, 0.7]$).

En el Anexo A se muestran tres histogramas de frecuencias de los tiempos de ejecución, para el 60, 75 y 90 % de la utilización total. La mayoría de las veces se obtuvieron tiempos bajos, por lo que los promedios mostrados en la tabla anterior son buenos estimados del tiempo de ejecución del algoritmo.

### 5.1.2. Tareas con utilizaciones entre 0.1 y 0.4

En este experimento, se consideraron tareas con utilizaciones variando entre 0.1 y 0.4. Para este, se generaron 1000 conjuntos de tareas para cada valor de la utilización total del sistema multiprocesador, variando entre el 50 y el 100 %. La Figura 5.2 muestra que la cota de utilización planificable para que un conjunto de tareas de estas características sea planificable bajo EDF usando esquema particionado en un sistema multiprocesador es estrictamente menor al 90 %. Aunque todos los conjuntos generados fueron planificados hasta el 90 % ($U_t = 3.6$) de la capacidad disponible, 90 % no puede ser una cota. El conjunto $\{0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4\}$ cuya utilización total es de 3.6 no puede ser planificado en su totalidad (bajo el modelo propuesto) en 4 procesadores: dos tareas de utilizaciones 0.4 son asignadas a cada

[1]El peor caso es cuando se alcanza la máxima utilización total, puesto que al no haber ninguna holgura ya que la utilización total coincide con la capacidad disponible, es más difícil acomodar las tareas en los procesadores.

procesador y queda una de utilización 0.4 que no puede ser asignada a ningún procesador, pues no puede ser dividida en dos.

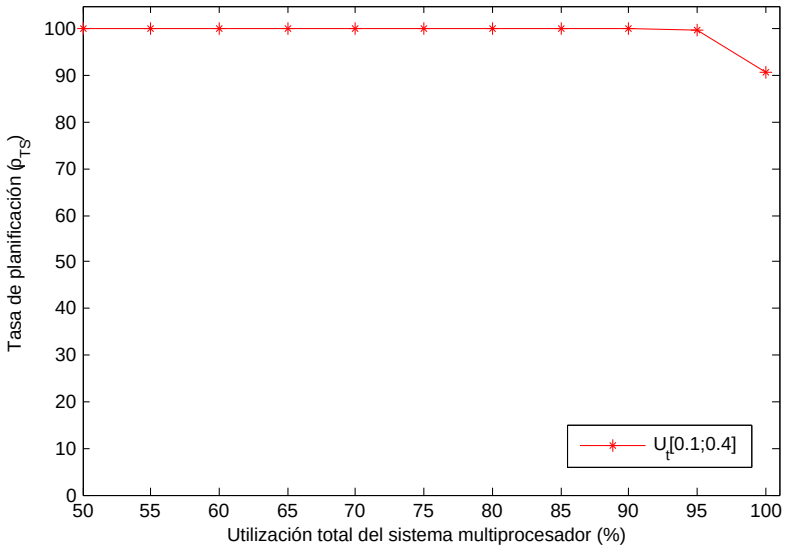


Figura 5.2: Tasa de planificación de los conjuntos de tareas (4 procesadores, $u_i \in [0.1, 0.4]$).

Para este tipo experimento, donde todas las tareas consumen poco tiempo de CPU, el algoritmo toma en promedio 844 segundos en terminar para el peor caso (15 minutos en promedio), para el 100 % de la utilización total. Análogamente, en el Anexo A se muestran algunos histogramas de frecuencias de los tiempos de ejecución para el 60, 75 y 90 % de la utilización total. Estos histogramas evidencian la validez de los promedios de los tiempos de ejecución de la siguiente Tabla:

| $U_t$ (%) | # iterac. | $t$ (seg.) |
|---|---|---|
| 50 | 13.4 | $8.5 \cdot 10^{-5}$ |
| 55 | 15.8 | $1.4 \cdot 10^{-4}$ |
| 60 | 18.4 | $2 \cdot 10^{-4}$ |
| 65 | 78 | $9 \cdot 10^{-4}$ |
| 70 | $3.7 \cdot 10^{3}$ | $3.4 \cdot 10^{-2}$ |
| 75 | $2.7 \cdot 10^{4}$ | 0.282 |
| 80 | $1.2 \cdot 10^{5}$ | 1.31 |
| 85 | $4.9 \cdot 10^{5}$ | 5.96 |
| 90 | $1.8 \cdot 10^{6}$ | 26 |
| 95 | $6 \cdot 10^{6}$ | 92 |
| 100 | $5.4 \cdot 10^{7}$ | 844 |

Tabla 5.2: Tiempo (en segundos) y número de iteraciones (4 procesadores, $u_i \in [0.1, 0.4]$).

### 5.1.3. Tareas con utilizaciones entre 0.1 y 0.3

En este experimento, se consideraron tareas con utilizaciones variando entre 0.1 y 0.3. Para este, se generaron 500 conjuntos de tareas para cada valor de la utilización total del sistema multiprocesador, variando entre el 50 y el 100 %. La tasa de planificación fue del 100 % para todos los valores de la utilización total del sistema multiprocesador (véase la siguiente Figura).

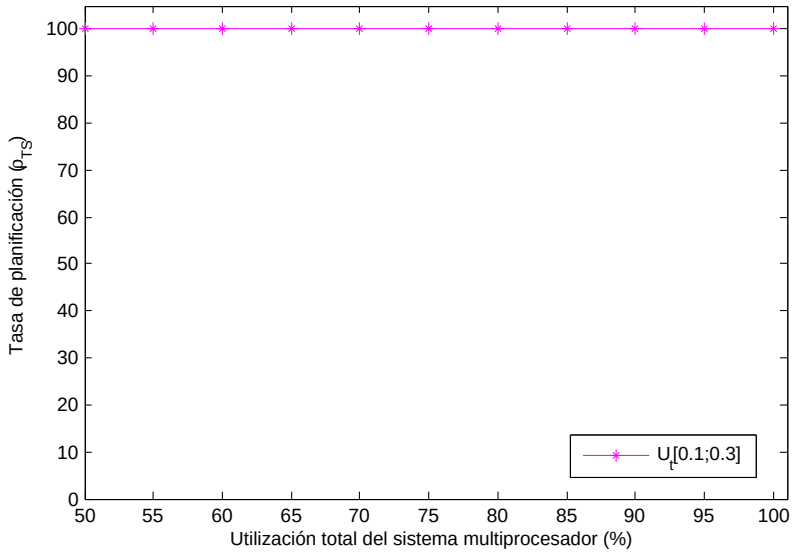


Figura 5.3: Tasa de planificación de los conjuntos de tareas (4 procesadores, $u_i \in [0.1, 0.3]$).

Esto no quiere decir que la cota de utilización planificable para que un conjunto de tareas de estas características sea planificable bajo EDF usando esquema particionado en un sistema multiprocesador sea del 100 %: el conjunto de 14 tareas, de las cuales 13 tienen utilización 0.3 y la otra tiene utilización 0.1 ($U_t = 4$) no es planificable bajo EDF usando esquema particionado bajo el modelo de tareas propuesto. Se asignarían 3 tareas de utilización 0.3 a cada procesador, con lo cual quedarían sin asignar una tarea de utilización 0.1 y una de 0.3. La primera puede ser asignada a cualquier procesador, pues cada uno tiene una capacidad restante de precisamente 0.1; pero la segunda no puede ser asignada a ninguno, pues no puede ser dividida en dos o más partes.

En este experimento, las tareas tienen aún una utilización más baja que en el experimento anterior. El algoritmo demora $1.37 \cdot 10^4$ segundos aproximadamente (tres horas y media) en terminar para el peor caso, es decir, para el 100 % de la utilización total. Al igual que en los experimentos anteriores, en el Anexo A se muestran tres histogramas de frecuencias para los tiempos de ejecución correspondientes al 60, 75 y 90 % de la utilización total, que evidencian que los promedios de los tiempos de

ejecución son válidos como un aproximado del tiempo de ejecución esperado.

| $U_t$ (%) | # iterac. | $t$ (seg.) |
|---|---|---|
| 50 | 19 | $2 \cdot 10^{-4}$ |
| 55 | 21.8 | $4.2 \cdot 10^{-4}$ |
| 60 | 24.5 | $4.1 \cdot 10^{-4}$ |
| 65 | $8.9 \cdot 10^{3}$ | $9.4 \cdot 10^{-2}$ |
| 70 | $9.6 \cdot 10^{4}$ | 1.08 |
| 75 | $8 \cdot 10^{5}$ | 10.82 |
| 80 | $2.2 \cdot 10^{6}$ | 30.5 |
| 85 | $2 \cdot 10^{7}$ | $3.5 \cdot 10^{2}$ |
| 90 | $5.2 \cdot 10^{7}$ | $9 \cdot 10^{2}$ |
| 95 | $1.9 \cdot 10^{8}$ | $3.6 \cdot 10^{3}$ |
| 100 | $6.77 \cdot 10^{8}$ | $1.37 \cdot 10^{4}$ |

Tabla 5.3: Tiempo (en segundos) y número de iteraciones (4 procesadores, $u_i \in [0.1, 0.3]$).

### 5.1.4. Tareas con utilizaciones entre 0.4 y 0.7

En este experimento, se consideraron tareas con utilizaciones variando entre 0.4 y 0.7 (sistema de tareas "pesadas"). Para este, se generaron aleatoriamente 1000 conjuntos de tareas para cada valor de la utilización total del sistema multiprocesador, variando entre el 50 y el 100 %.

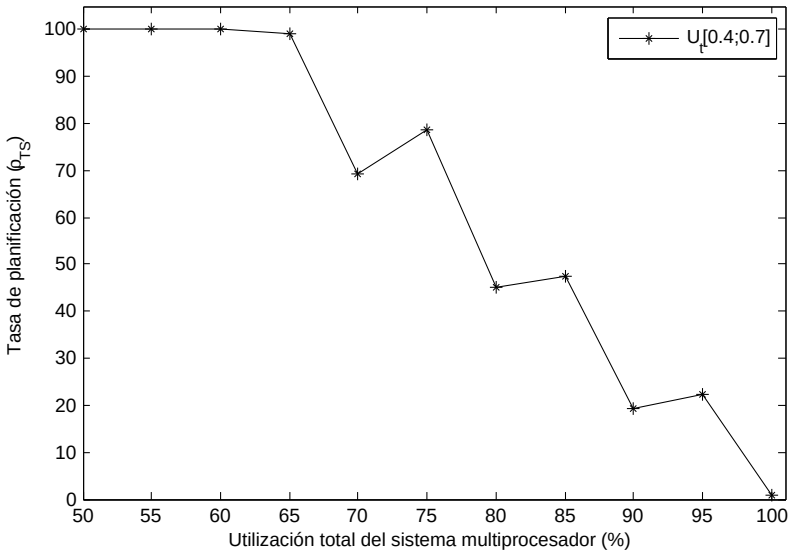


Figura 5.4: Tasa de planificación de los conjuntos de tareas (4 procesadores, $u_i \in [0.4, 0.7]$).

Se puede conjeturar que la cota de utilización planificable para que un conjunto de tareas de estas características sea planificable bajo EDF usando esquema particionado en un sistema multiprocesador es muy baja, quizás del 60 % o aún menor, como lo muestra la Figura 5.4.

Para este experimento, donde todas las tareas consumen mucho tiempo de CPU, los tiempos son muy bajos, lo cual era de esperarse, pues mientras más grandes son las utilizaciones de las tareas, menos variables hay. Además, muchas veces el filtrado eliminó tareas que no se podían asignar debido a sus grandes utilizaciones. En este experimento, el algoritmo toma en promedio 0.13 segundos en terminar para el peor caso, es decir, para el 100 % de la utilización total, como muestra la siguiente Tabla:

| $U_t$ ( %) | # iterac. | $t$ (seg.) |
|---|---|---|
| 50 | 1 | $2.03 \cdot 10^{-6}$ |
| 55 | 1.17 | $2.4 \cdot 10^{-6}$ |
| 60 | 2.4 | $5.14 \cdot 10^{-6}$ |
| 65 | 3.8 | $9.3 \cdot 10^{-6}$ |
| 70 | 3.75 | $1.27 \cdot 10^{-5}$ |
| 75 | 5.67 | $2.3 \cdot 10^{-5}$ |
| 80 | $5 \cdot 10^{2}$ | $2.2 \cdot 10^{-3}$ |
| 85 | $2.7 \cdot 10^{2}$ | $1.2 \cdot 10^{-3}$ |
| 90 | $4.4 \cdot 10^{3}$ | $2.26 \cdot 10^{-2}$ |
| 95 | $3.4 \cdot 10^{3}$ | $1.8 \cdot 10^{-2}$ |
| 100 | $2.17 \cdot 10^{4}$ | 0.131 |

Tabla 5.4: Tiempo (en segundos) y número de iteraciones (4 procesadores, $u_i \in [0.4, 0.7]$).

Los histogramas de frecuencia del Anexo A para los tiempos de ejecución, correspondientes al 60, 75 y 90 % de la utilización total de este experimento muestran que es muy poco probable que los tiempos de ejecución del algoritmo sobrepasen mucho el promedio, por lo que los valores promedio de la tabla anterior son válidos como valores aproximados de los tiempos de ejecución.

### 5.1.5. Tareas con utilizaciones entre 0.05 y 0.2

En este experimento, se consideraron tareas con utilizaciones variando entre 0.05 y 0.2. Se generaron 500 conjuntos de tareas para cada valor de la utilización total del sistema multiprocesador, variando entre el 50 y el 70 %. Para este experimento, sólo se llegó a una utilización total del 70 %, pues los tiempos de respuesta aumentaron demasiado rápidamente, como se verá en el ploteo de la siguiente sección. Esto era de esperarse, pues como las utilizaciones de las tareas son tan pequeñas, el algoritmo debe trabajar con muchas más variables que en los experimentos anteriores. Hasta el 70 % de la utilización total, la tasa de planificación fue del 100 %. Los tiempos de

ejecución se muestran en la siguiente Tabla. Según los histogramas de frecuencia del Anexo A correspondientes a este experimento, es muy probable que se obtenga un tiempo de ejecución cercano al promedio y no muy superior a él.

| $U_t$ (%) | # iterac. | $t$ (seg.) |
|---|---|---|
| 50 | 35.4 | $9 \cdot 10^{-4}$ |
| 55 | 40.4 | $1.4 \cdot 10^{-3}$ |
| 60 | $1.5 \cdot 10^{5}$ | 2.4 |
| 65 | $9.8 \cdot 10^{7}$ | $2.1 \cdot 10^{3}$ |
| 70 | $1.07 \cdot 10^{8}$ | $1.15 \cdot 10^{4}$ |

Tabla 5.5: Tiempo (en segundos) y número de iteraciones (4 procesadores, $u_i \in [0.05, 0.2]$).

### 5.1.6. Comparación de tiempos de los experimentos realizados con 4 procesadores

En esta sección se presentan gráficas comparativas de los cinco experimentos realizados con 4 procesadores. Según se puede apreciar, los tiempos de ejecución aumentan muy rápidamente a medida que aumenta la cantidad de tareas pequeñas en el sistema. Esto era de esperarse, pues por cada tarea en el sistema, hay $m$ variables en el modelo matemático, por lo tanto mientras más tareas hay, con más variables debe trabajar el algoritmo.

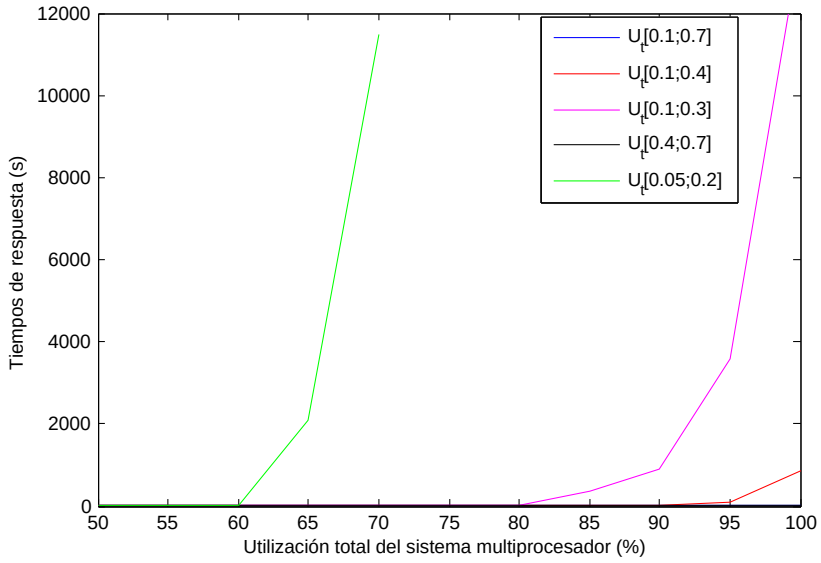


Figura 5.5: Tiempos de respuesta para los cinco experimentos realizados con 4 procesadores.

La Figura 5.6 muestra la sección inferior de la Figura 5.5 de manera que se pueda observar más claramente la comparación de los tiempos de ejecución de los cinco experimentos realizados. Los tiempos de ejecución aumentan exponencialmente, y este crecimiento se ve más marcado en los experimentos que restringen a las tareas a tener utilizaciones muy pequeñas.

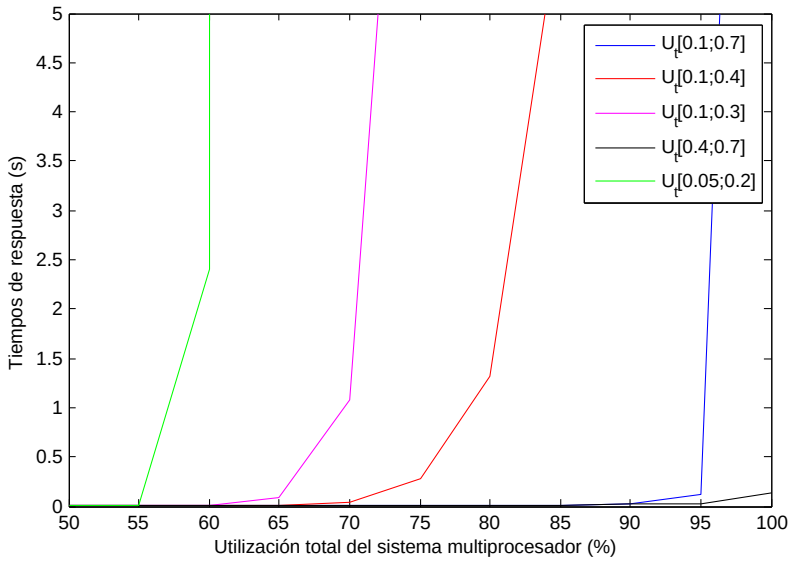


Figura 5.6: Sección inferior de la Figura 5.5 con un cambio de escala en el eje $Y$.

## 5.2. 6 procesadores

### 5.2.1. Tareas con utilizaciones entre 0.1 y 0.7

En este experimento, se consideraron 6 procesadores, y se generaron aleatoriamente conjuntos de tareas con utilizaciones variando entre 0.1 y 0.7. Se generaron 1000 conjuntos de tareas para cada valor de la utilización total del sistema multiprocesador, variando entre el 50 y el 100 %. La siguiente Figura sugiere que la cota de utilización planificable para que un conjunto de tareas de estas características sea planificable bajo EDF usando esquema particionado en un sistema multiprocesador es bastante baja, quizás entre el el 65 y el 70 %.

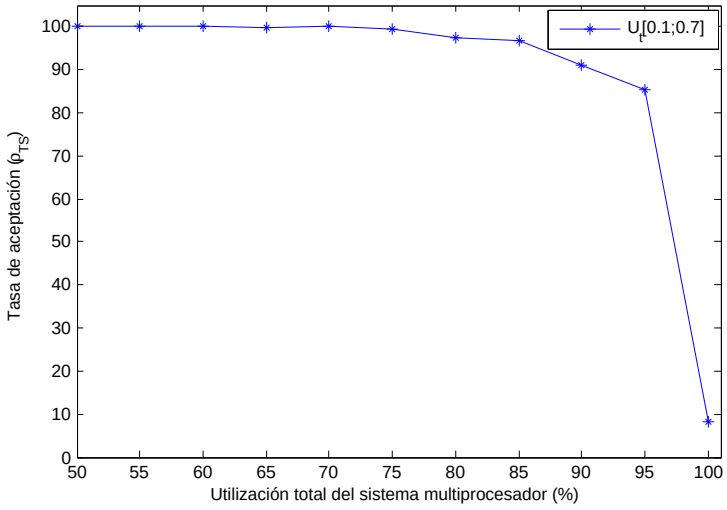


Figura 5.7: Tasa de planificación de los conjuntos de tareas (6 procesadores, $u_i \in [0.1, 0.7]$).

En lo que respecta a los tiempos de ejecución, se mantienen bastante bajos hasta que se alcanza el 85 % de la utilización total. Los histogramas de frecuencia del Anexo A, correspondientes a este experimento, muestran que es muy probable que se obtengan tiempos de ejecución cercanos a los promedios de la siguiente Tabla:

| $U_t$ (%) | # iterac. | $t$ (seg.) |
|---|---|---|
| 50 | 10.5 | $8.96 \cdot 10^{-5}$ |
| 55 | 14 | $1.5 \cdot 10^{-4}$ |
| 60 | 95.4 | $1.57 \cdot 10^{-3}$ |
| 65 | 156.3 | $2.4 \cdot 10^{-3}$ |
| 70 | $1.76 \cdot 10^{4}$ | 0.28 |
| 75 | $1.06 \cdot 10^{5}$ | 1.83 |
| 80 | $2.67 \cdot 10^{5}$ | 4.48 |
| 85 | $1.01 \cdot 10^{6}$ | 18.08 |
| 90 | $6.65 \cdot 10^{6}$ | $1.16 \cdot 10^{2}$ |
| 95 | $8.3 \cdot 10^{7}$ | $1.45 \cdot 10^{3}$ |
| 100 | $1.8 \cdot 10^{9}$ | $5.2 \cdot 10^{4}$ |

Tabla 5.6: Tiempo (en segundos) y número de iteraciones (6 procesadores, $u_i \in [0.1, 0.7]$).

### 5.2.2. Tareas con utilizaciones entre 0.4 y 0.7

En este experimento, se consideraron 6 procesadores, y se generaron aleatoriamente conjuntos de tareas que consumen mucho tiempo de CPU, con utilizaciones

variando entre 0.4 y 0.7. Se generaron 1000 conjuntos de tareas para cada valor de la utilización total del sistema multiprocesador, variando entre el 50 y el 100 %. Según se aprecia en la siguiente Figura, la tasa de planificación decrece rápidamente, pues como las tareas tienen utilizaciones grandes y no se pueden dividir, a medida que aumente la utilización total del sistema, será más difícil poderlas planificar. La cota de utilización planificable para que un conjunto de tareas de estas características sea planificable bajo EDF usando esquema particionado en un sistema multiprocesador es muy baja, alrededor del 55 %.

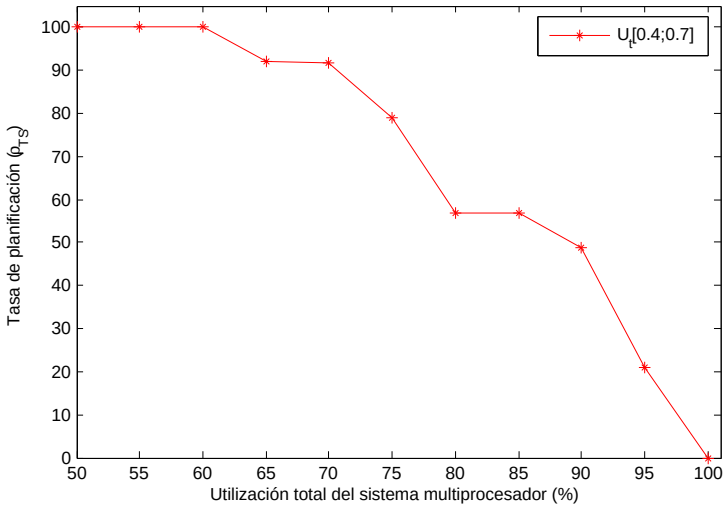


Figura 5.8: Tasa de planificación de los conjuntos de tareas (6 procesadores, $u_i \in [0.4, 0.7]$).

| $U_t$ ( %) | # iterac. | $t$ (seg.) |
|---|---|---|
| 50 | 1.01 | $3.5 \cdot 10^{-6}$ |
| 55 | 2.2 | $1.01 \cdot 10^{-5}$ |
| 60 | 5.66 | $3 \cdot 10^{-5}$ |
| 65 | 8.08 | $6.4 \cdot 10^{-5}$ |
| 70 | $2.3 \cdot 10^{3}$ | $2.2 \cdot 10^{-2}$ |
| 75 | $1.07 \cdot 10^{4}$ | 0.1 |
| 80 | $1.9 \cdot 10^{5}$ | 1.99 |
| 85 | $1.22 \cdot 10^{6}$ | 14.3 |
| 90 | $1.58 \cdot 10^{6}$ | 21.59 |
| 95 | $7.25 \cdot 10^{6}$ | 96.5 |
| 100 | $5.3 \cdot 10^{7}$ | $8 \cdot 10^{2}$ |

Tabla 5.7: Tiempo (en segundos) y número de iteraciones (6 procesadores, $u_i \in [0.4, 0.7]$).

Los tiempos de ejecución se mantienen bastante bajos hasta que se alcanza el 90 % de la utilización total. Al igual que en los experimentos anteriores, los histogramas de frecuencia del Anexo A para los tiempos de ejecución, correspondientes al 60, 75 y 90 % de la utilización total de este experimento muestran la validez de los valores promedio de la tabla anterior como valores aproximados de los tiempos de ejecución.

### 5.2.3. Comparación de tiempos de los experimentos realizados con 6 procesadores

A continuación se muestra la gráfica comparativa de los tiempos de ejecución de los dos experimentos realizados para 6 procesadores. Los tiempos de ejecución también aumentan exponencialmente, y el crecimiento es mayor en el caso que las utilizaciones de las tareas están en el intervalo $[0.1; 0.7]$.

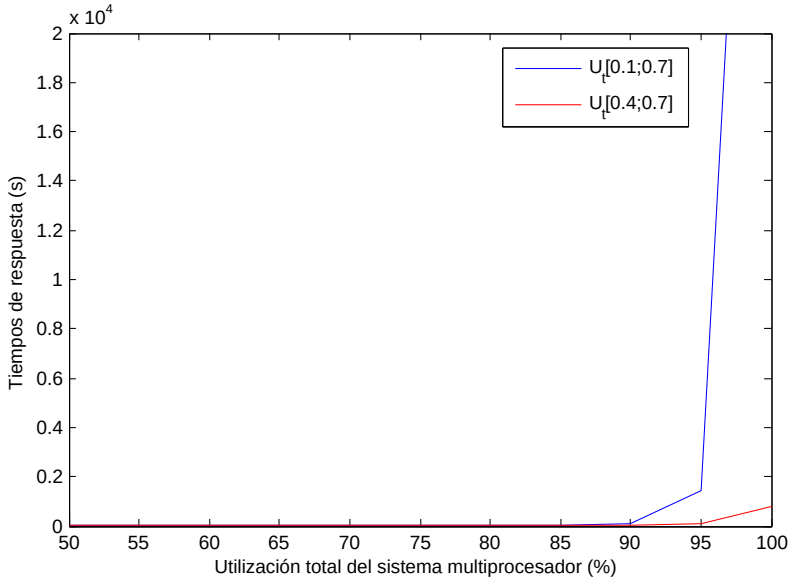


Figura 5.9: Tiempos de respuesta para los cuatro experimentos realizados con 6 procesadores.

Aunque no es propósito de este trabajo de tesis realizar una comparación de algoritmos, no está de más comentar que en el área de investigación de operaciones existen benchmarks clásicos para evaluar el desempeño de algoritmos de planificación, para problemas básicos de planificación, como por ejemplo los benchmarks de Taillard [Tai93]). Sin embargo, estos benchmarks no se ajustan a los propósitos de este trabajo, debido a las diferencias entre la planificación en investigación de operaciones y la planificación de los sistemas de tiempo real. Por ejemplo, en investigación de operaciones no existen plazos ni tiempos de activación para las tareas y usualmente el objetivo es minimizar el tiempo de ejecución total. En la planificación de tiempo

real, esto no constituye un objetivo, sino que se cumplan los plazos de las tareas. En la planificación de tiempo real no existen benchmarks estándares como los de Taillard. Se usan, en cambio, conjuntos de tareas generados aleatoriamente con ciertas características estadísticamente apropiadas.

# Capítulo 6

# Conclusiones

La dirección principal de la investigación en la planificación multiprocesador de tiempo real ha sido desarrollar algoritmos heurísticos eficientes que encuentren soluciones aproximadas en tiempo polinomial, y se ha estudiado muy poco la posibilidad de aplicar técnicas exactas de optimización para obtener la planificación óptima en este tipo de sistemas. Existen algunos trabajos previos que se enfocan en hallar una solución óptima para la planificación de un conjunto de tareas bajo el esquema particionado, pero no hay resultados experimentales disponibles para el modelo de tareas aquí propuesto.

Para problemas de tamaño pequeño propio de muchos sistemas de tiempo real, puede ser una opción factible utilizar algoritmos exactos para encontrar una asignación óptima fuera de línea de un conjunto de tareas en un sistema multiprocesador. Con este trabajo de tesis, se evaluó, con resultados experimentales, la factibilidad de la aplicación de un algoritmo exacto de optimización entera para hallar la solución óptima al problema de la planificación multiprocesador bajo un determinado modelo de tareas.

Los resultados experimentales que aquí se presentan para cuatro y seis procesadores son válidos. Puesto que el problema de asignación multiprocesador se clasifica como *NP-hard*, no se conoce ningún algoritmo de complejidad polinomial que lo resuelva. Para aplicaciones embebidas y de tiempo real, cuatro ó seis procesadores es un límite práctico, debido a los costos, y restricciones de energía y espacio típicos de las plataformas de hardware de este tipo de sistemas. Por consiguiente, los resultados experimentales de este trabajo de tesis proveen evidencias que demuestran que, para algunas aplicaciones de tiempo real con configuraciones de número de procesadores de tamaño típico, un algoritmo exacto fuera de línea (como el que se propone), puede ser usado en vez de un algoritmo aproximado eficiente.

Aplicando la versión simplificada del algoritmo de Balas dada por Geoffrion, conjuntamente con las modificaciones expuestas en el Capítulo anterior, para adaptar el algoritmo al problema específico de planificación multiprocesador, se obtiene un mejor desempeño, en comparación con cualquier algoritmo heurístico, pues el método

que aquí se propone siempre encuentra la planificación óptima si ésta existe. La única desventaja de este enfoque es que puede tomar mucho tiempo antes de obtenerse la planificación óptima.

Para los sistemas de tareas que consumen mucho tiempo de CPU ($u_i \in [0.4; 0.7]\ \forall i$), para los sistemas que son una mezcla de tareas que consumen mucho y poco tiempo de CPU ($u_i \in [0.1; 0.7]\ \forall i$) y para algunos sistemas que consumen poco tiempo de CPU ($u_i \in [0.1; 0.4]\ \forall i$) es perfectamente posible utilizar este algoritmo fuera de línea para hallar la planificación óptima, debido a que los tiempos de ejecución son bajos teniendo en cuenta la complejidad del problema que se está resolviendo. Para otro tipo de sistemas donde las tareas tienen utilizaciones más pequeñas ($u_i \in [0.1; 0.3]\ \forall i$), hasta el 80 % de la utilización total del sistema multiprocesador los tiempos se mantienen razonablemente bajos. Para el experimento realizado con sistemas de tareas cuyas utilizaciones son muy pequeñas ($u_i \in [0.05; 0.2]\ \forall i$), los tiempos aumentaron demasiado, pues mientras más pequeñas son las utilizaciones de las tareas, más tareas hay, y por lo tanto, el algoritmo trabaja con muchas más variables (por cada tarea, hay $m$ variables). Esto demuestra que para este tipo de sistemas no resulta factible aplicar un algoritmo exacto, al menos sin modificar el algoritmo para reducir los tiempos de cómputo.

Para el experimento realizado con seis procesadores, en el que se consideraron sistemas con una mezcla de tareas que consumen mucho y poco tiempo de CPU ($u_i \in [0.1; 0.7]\ \forall i$), los tiempos se mantienen bajos hasta que se alcanza el 85 % de la utilización total. En el caso del experimento realizado con seis procesadores que considera sistemas de tareas "pesadas", esto es, que consumen mucho tiempo de CPU, ($u_i \in [0.4; 0.7]\ \forall i$), los tiempos se mantienen bajos hasta que se alcanza el 90 % de la utilización total.

Además, los histogramas de frecuencias del Anexo A demuestran la validez de los tiempos promedios de ejecución como valores aproximados del tiempo de ejecución del algoritmo para cada porcentaje de utilización total. Aunque no se han realizado estudios con el objetivo de hallar formalmente la distribución por la cual se rigen los tiempos de respuesta, a simple vista se puede decir que muchos de estos histogramas se asemejan a una distribución exponencial negativa, que es la que generalmente rige los tiempos de respuesta de servidores y otros procesos en el tiempo. En este tipo de distribución, hay una probabilidad muy alta de que el tiempo de respuesta de una ejecución del algoritmo esté próximo al tiempo promedio de respuesta.

## 6.1. Contribuciones principales

Las contribuciones principales de este trabajo de tesis son las siguientes:

1. Se caracteriza el problema de la planificación multiprocesador con esquema particionado bajo el modelo de tareas propuesto, como un problema de programación lineal entera binaria.

2. Se realizan modificaciones a un algoritmo clásico de optimización entera, usando conocimiento del problema de la planificación multiprocesador, para adaptarlo a éste.

3. Se ofrecen resultados experimentales de la aplicación de un algoritmo exacto de optimización entera, para hallar la planificación óptima bajo el algoritmo EDF usando esquema particionado, en un sistema multiprocesador.

4. Se obtienen valores aproximados para las cotas de utilización para la planificabilidad de un conjunto de tareas[1] bajo EDF usando esquema particionado.

## 6.2. Trabajo futuro

El algoritmo propuesto en este trabajo de tesis aún puede ser modificado de varias maneras para reducir el tiempo de ejecución. Este es el objetivo principal, pues el algoritmo encuentra una planificación óptima para el modelo de tareas propuesto en la Sección 4.1 del Capítulo 4.

Una posible dirección de trabajo puede ser la aplicación de la mejora propuesta por Geoffrion en [Geo69]. En este artículo, se propone la introducción de restricciones de sustitución, las cuales fueron expuestas en la Sección 4.7. Estas restricciones encapsulan la información del problema original de programación entera y al aplicar los criterios de sondeo a un menor número de restricciones, se aceleraría el proceso de poda. En problemas de mochila multidimensional, la introducción de este tipo de restricciones en cada iteración del algoritmo de la Figura 4.5, conlleva a tiempos de ejecución que parecen incrementarse como la tercera potencia del número de variables [Geo69]. El problema de la asignación en la planificación multiprocesador no es un problema de mochila multidimensional, sino de mochila múltiple (esto implica muchas más restricciones). Por esta razón, restaría evaluar cuánto reduciría los tiempos de ejecución el uso de estas restricciones aplicando el algoritmo de la Figura 4.6.

Otras posibles modificaciones al algoritmo pueden ser:

- Resolver el problema relajado asociado al problema original, para comenzar con una solución inicial que esté dada por los valores enteros más cercanos a la solución óptima de la versión continua del problema original. Esta solución inicial, (llamada *LP-start*) ha probado ser efectiva en algunos problemas [Geo69].

- Usar otros criterios de sondeo que sean más poderosos que el criterio de Balas para el problema de la asignación multiprocesador. Algunos criterios de sondeo se describen en [Pet67] conjuntamente con una comparación de tiempos de cómputo en varios proyectos de investigación y desarrollo.

[1] Bajo el modelo de tareas propuesto.

Luego que se haya implementado la introducción de las restricciones de sustitución, se puede comparar entre la propuesta original de Geoffrion [Geo69] y las siguientes modificaciones:

- Introducir las restricciones de sustitución cada cierto número de iteraciones. Se reportan trabajos donde el tiempo de ejecución fue menor cuando no se introducían las restricciones en cada iteración [Geo69].
- Utilizar un algoritmo de punto interior (e.g. el algoritmo de Mehrotra descrito en la Sección 3.4.1 del Capítulo 3) en lugar del método Simplex para resolver el (LP) necesario para hallar los coeficientes de la restricción de sustitución, en caso que se vaya a trabajar con problemas grandes.

Se podría pensar también en la paralelización de los criterios de sondeo del algoritmo, para acelerar el proceso de poda y así reducir el tiempo de cómputo. Además, se puede aumentar el número de experimentos y de esta forma, estimar de manera más exacta la utilización promedio planificable de los conjuntos de tareas con las características descritas en el Capítulo 5, bajo EDF usando esquema particionado en un sistema multiprocesador.

La desventaja principal del enfoque exacto propuesto en este trabajo de tesis es que puede tomar mucho tiempo encontrar la planificación óptima. Es por esta razón que se ha trabajado intensamente en buscar algoritmos heurísticos eficientes que brindan una "buena "solución en tiempo polinomial. Podría pensarse en un enfoque híbrido entre alguna heurística (que se utilizaría por ejemplo en hallar una solución inicial) y un enfoque exacto como el que aquí se describe. También se pueden estudiar otros algoritmos de optimización entera que pudieran tener mejor desempeño para el problema específico de la planificación multiprocesador, por ejemplo los basados en las técnicas de los planos cortantes descritas en el Capítulo 3.

# Anexos

# Anexo A

En este apéndice se muestran tres histogramas de frecuencia de los tiempos de ejecución para cada experimento realizado, generalmente para el 60, 75 y 90 % de la utilización total. Por razones de espacio solo se muestran tres histogramas, pero para los demás porcentajes de la utilización total, los tiempos de ejecución siguen generalmente el mismo comportamiento. El eje X que corresponde a los tiempos de ejecución, está dividido en 10 clases, y el eje Y corresponde a la cantidad de experimentos que clasificaron en cada clase.

## A.1. Experimento 1: 4 procesadores, utilizaciones entre 0.1 y 0.7

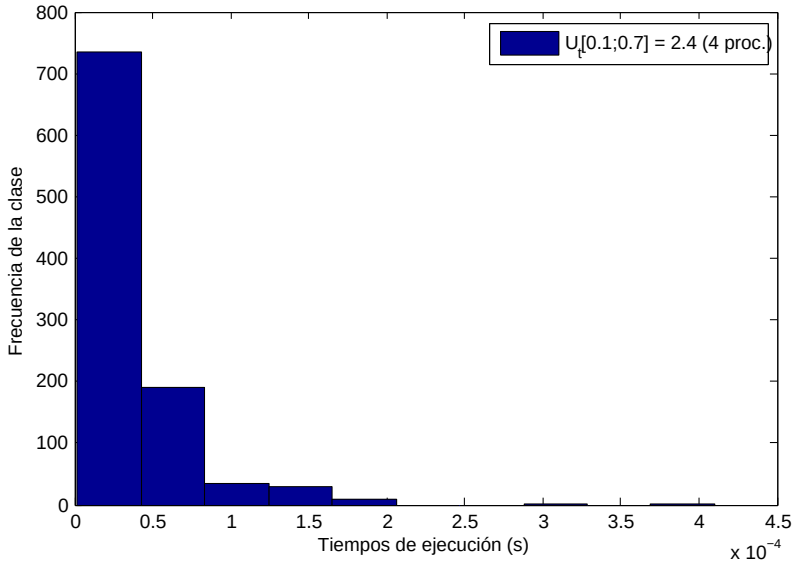


Figura A.1: Histograma de frecuencias de los tiempos de ejecución (4 procesadores, $u_i \in [0.1, 0.7]$, $U_T = 60\,\%$).

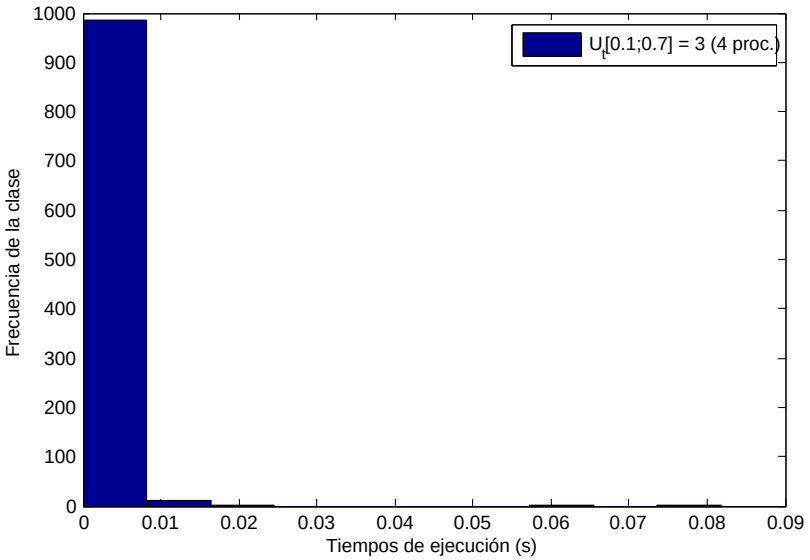


Figura A.2: Histograma de frecuencias de los tiempos de ejecución (4 procesadores, $u_i \in [0.1, 0.7]$, $U_T = 75\,\%$).

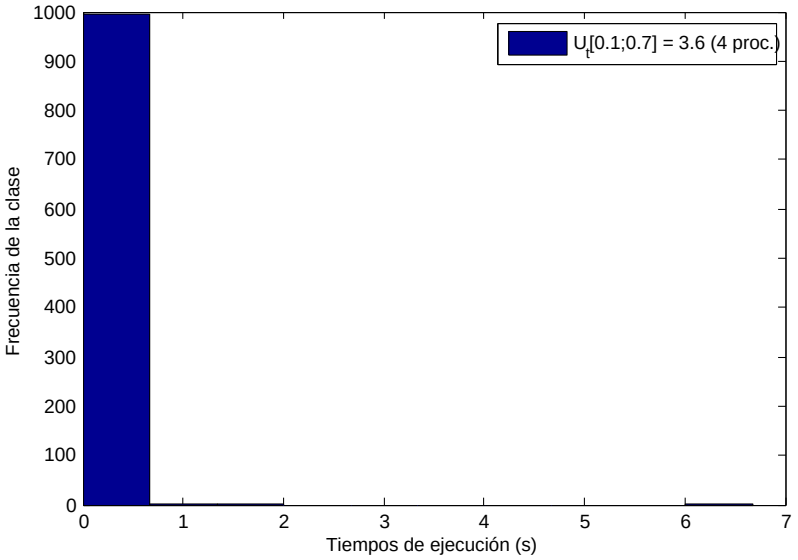


Figura A.3: Histograma de frecuencias de los tiempos de ejecución (4 procesadores, $u_i \in [0.1, 0.7]$, $U_T = 90\,\%$).

## A.2. Experimento 2: 4 procesadores, utilizaciones entre 0.1 y 0.4

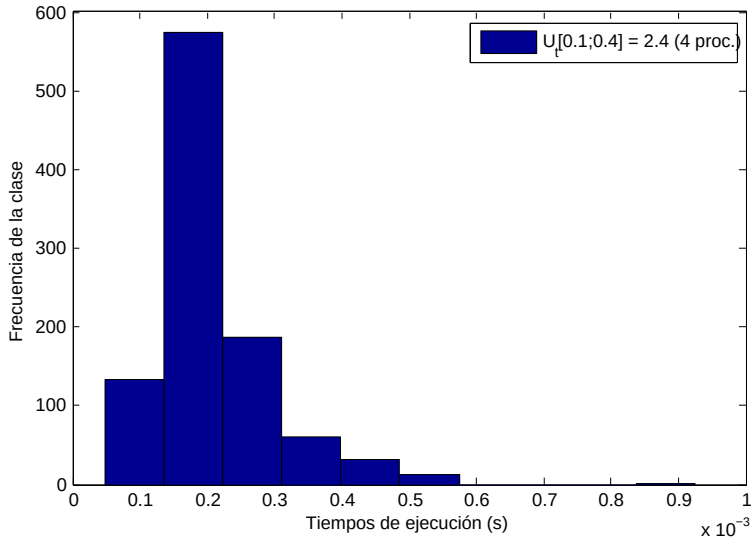


Figura A.4: Histograma de frecuencias de los tiempos de ejecución (4 procesadores, $u_i \in [0.1, 0.4]$, $U_T = 60\,\%$).

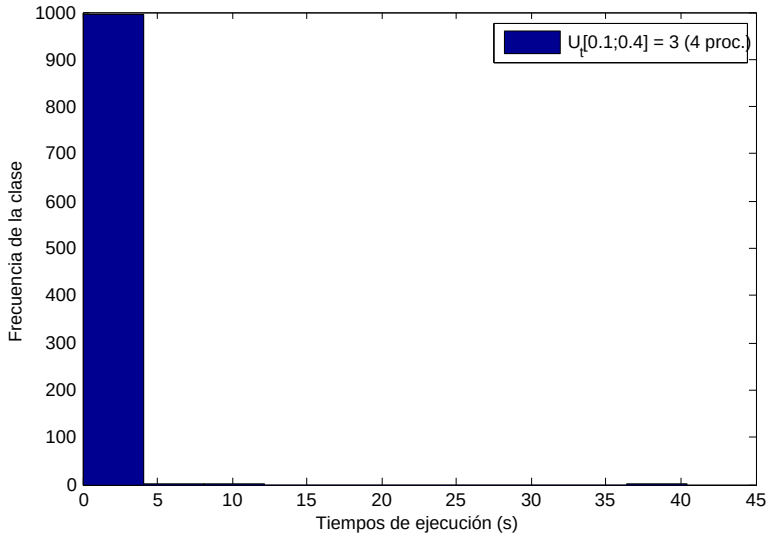


Figura A.5: Histograma de frecuencias de los tiempos de ejecución (4 procesadores, $u_i \in [0.1, 0.4]$, $U_T = 75\,\%$).

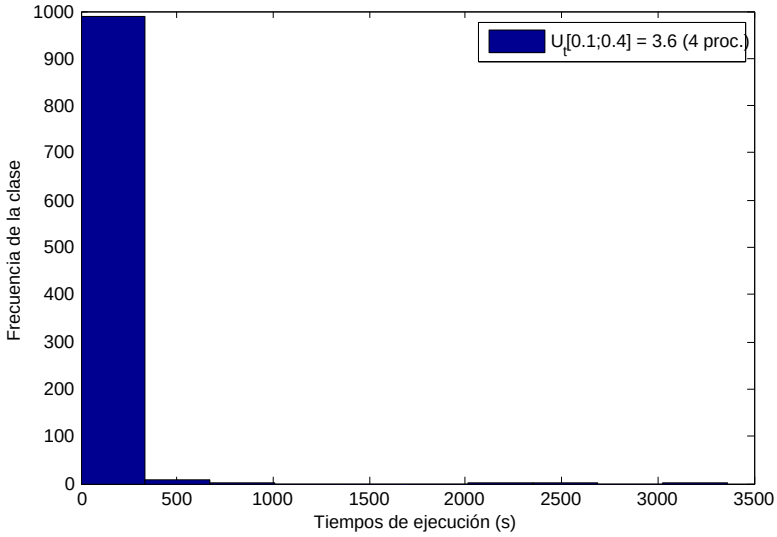


Figura A.6: Histograma de frecuencias de los tiempos de ejecución (4 procesadores, $u_i \in [0.1, 0.4]$, $U_T = 90\,\%$).

## A.3. Experimento 3: 4 procesadores, utilizaciones entre 0.1 y 0.3

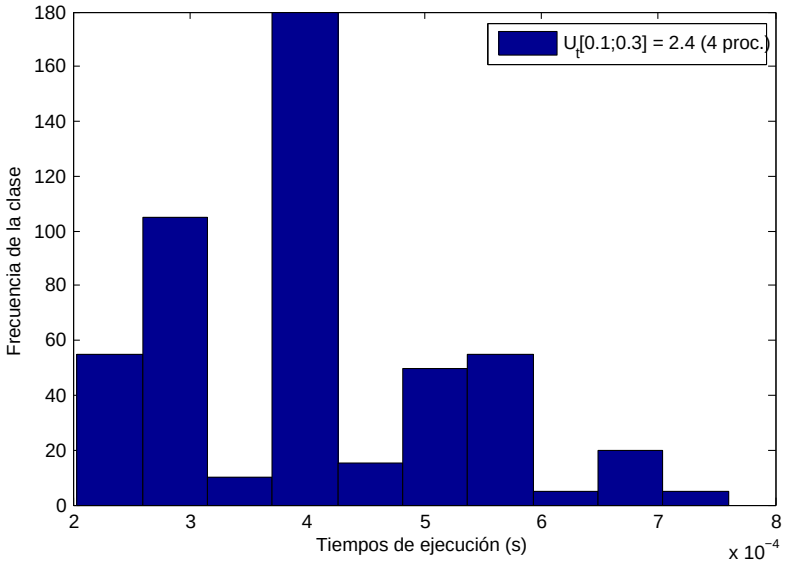


Figura A.7: Histograma de frecuencias de los tiempos de ejecución (4 procesadores, $u_i \in [0.1, 0.3]$, $U_T = 60\,\%$).

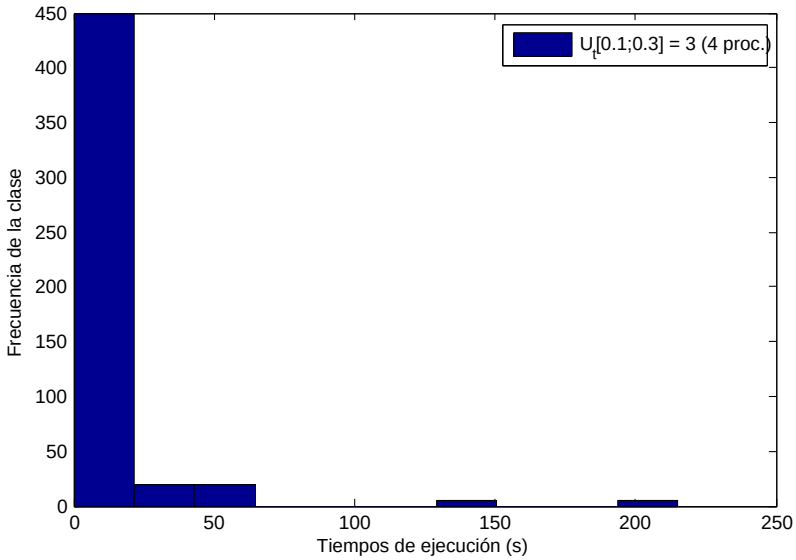


Figura A.8: Histograma de frecuencias de los tiempos de ejecución (4 procesadores, $u_i \in [0.1, 0.3]$, $U_T = 75\,\%$).

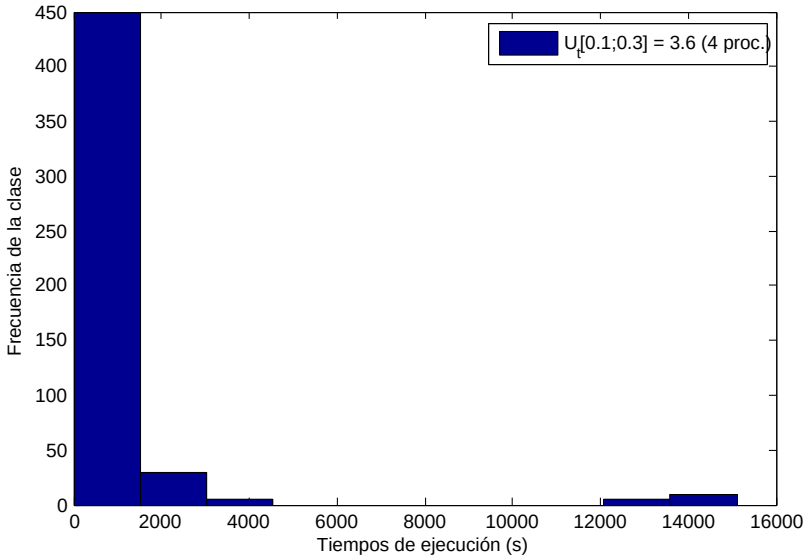


Figura A.9: Histograma de frecuencias de los tiempos de ejecución (4 procesadores, $u_i \in [0.1, 0.3]$, $U_T = 90\,\%$).

## A.4. Experimento 4: 4 procesadores, utilizaciones entre 0.4 y 0.7

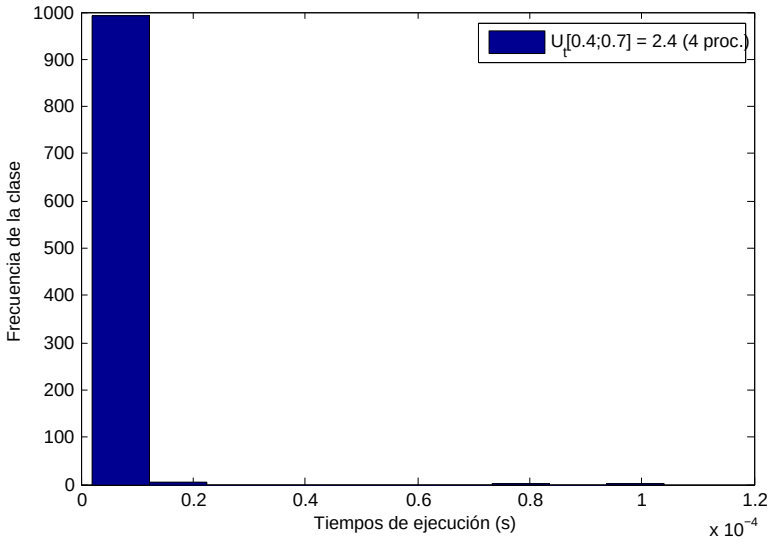


Figura A.10: Histograma de frecuencias de los tiempos de ejecución (4 procesadores, $u_i \in [0.4, 0.7]$, $U_T = 60\,\%$).

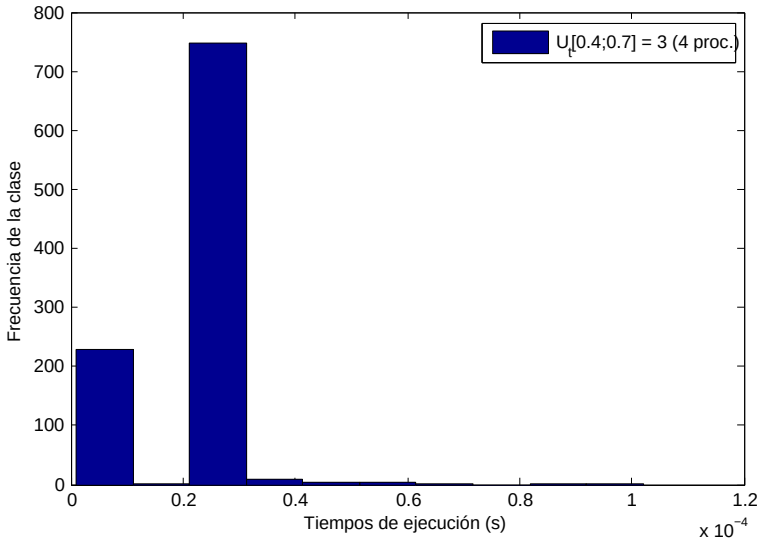


Figura A.11: Histograma de frecuencias de los tiempos de ejecución (4 procesadores, $u_i \in [0.4, 0.7]$, $U_T = 75\,\%$).

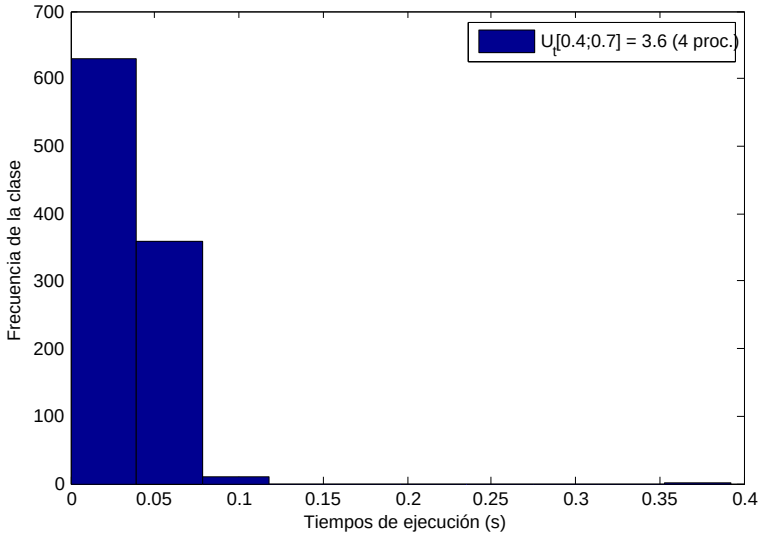


Figura A.12: Histograma de frecuencias de los tiempos de ejecución (4 procesadores, $u_i \in [0.4, 0.7]$, $U_T = 90\,\%$).

## A.5. Experimento 5: 4 procesadores, utilizaciones entre 0.05 y 0.2

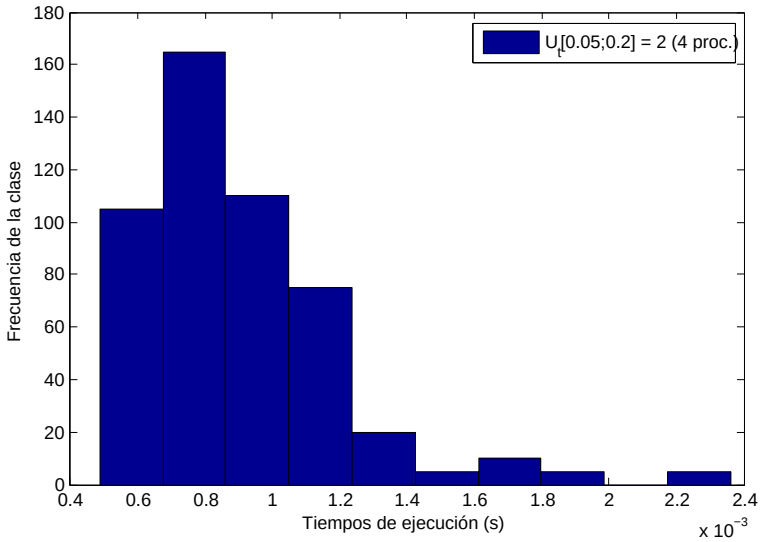


Figura A.13: Histograma de frecuencias de los tiempos de ejecución (4 procesadores, $u_i \in [0.05, 0.2]$, $U_T = 50\,\%$).

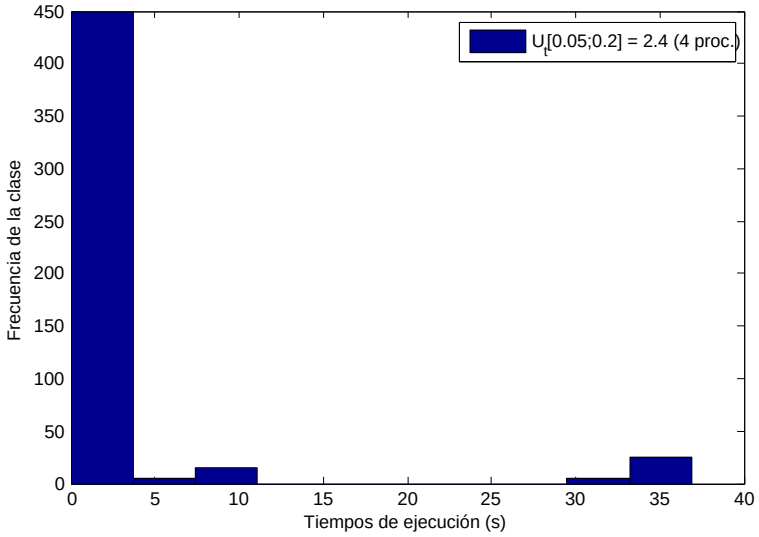


Figura A.14: Histograma de frecuencias de los tiempos de ejecución (4 procesadores, $u_i \in [0.05, 0.2]$, $U_T = 60\,\%$).

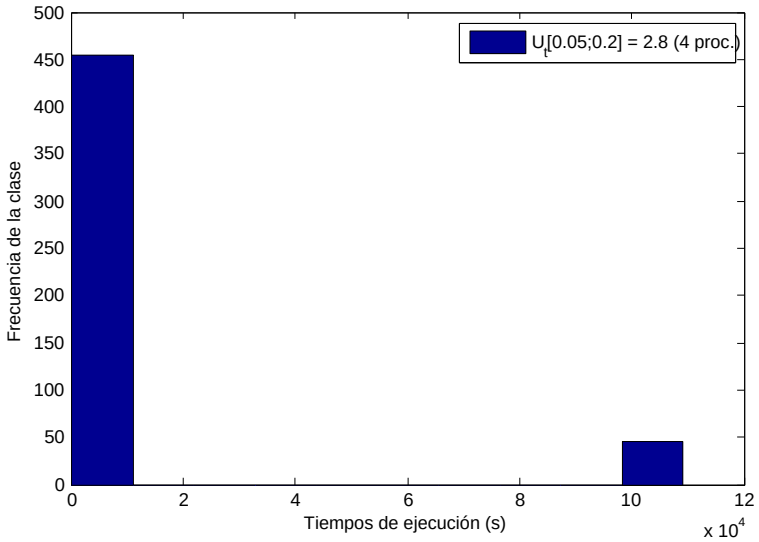


Figura A.15: Histograma de frecuencias de los tiempos de ejecución (4 procesadores, $u_i \in [0.05, 0.2]$, $U_T = 70\,\%$).

## A.6. Experimento 6: 6 procesadores, utilizaciones entre 0.1 y 0.7

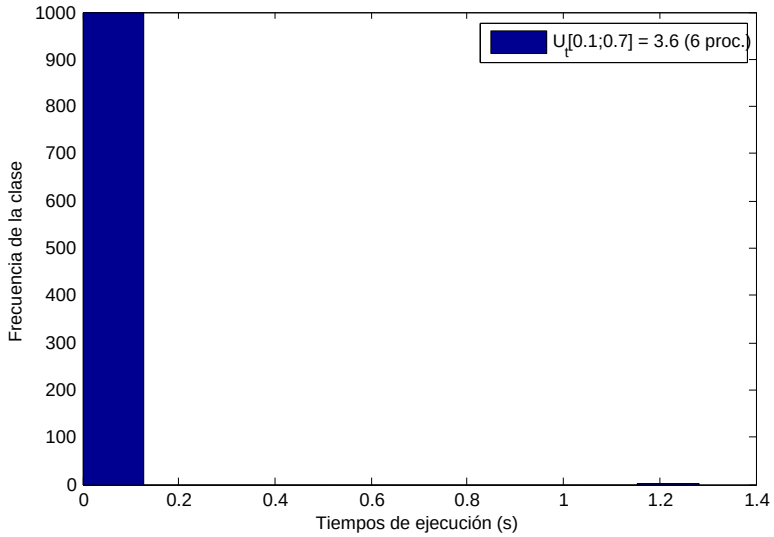


Figura A.16: Histograma de frecuencias de los tiempos de ejecución (6 procesadores, $u_i \in [0.1, 0.7]$, $U_T = 60\,\%$).

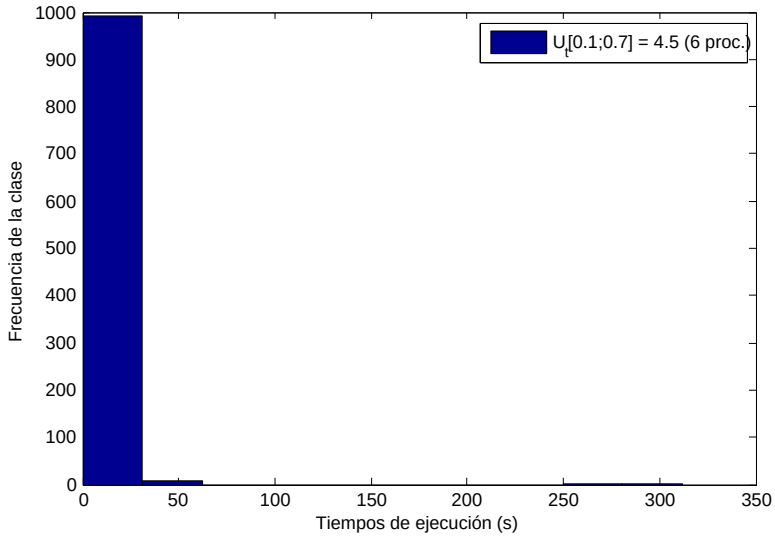


Figura A.17: Histograma de frecuencias de los tiempos de ejecución (6 procesadores, $u_i \in [0.1, 0.7]$, $U_T = 75\,\%$).

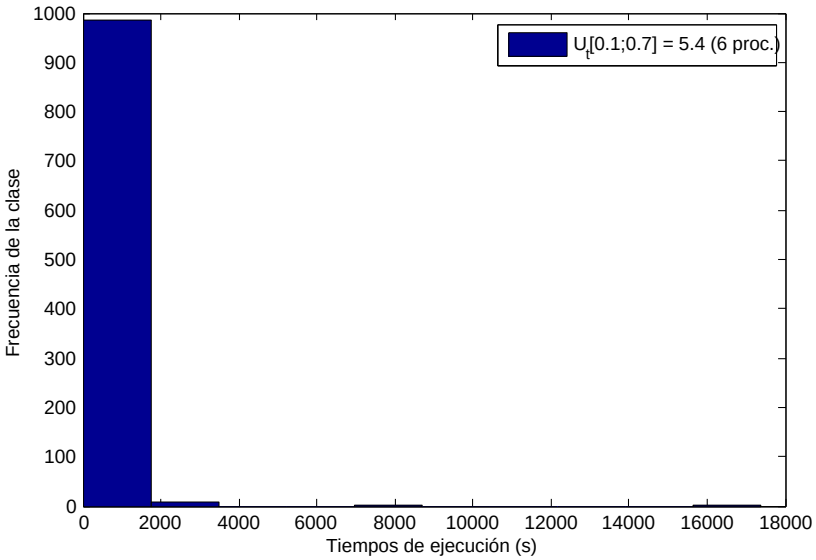


Figura A.18: Histograma de frecuencias de los tiempos de ejecución (6 procesadores, $u_i \in [0.1, 0.7]$, $U_T = 90\,\%$).

## A.7. Experimento 7: 6 procesadores, utilizaciones entre 0.4 y 0.7

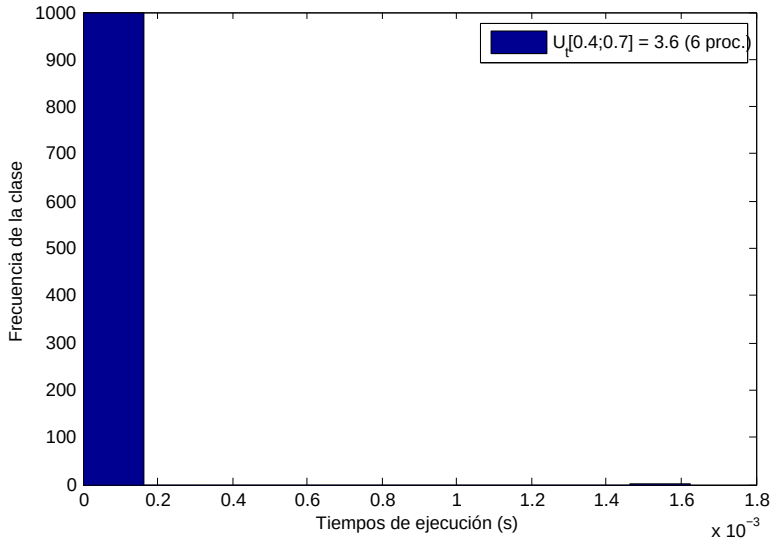


Figura A.19: Histograma de frecuencias de los tiempos de ejecución (6 procesadores, $u_i \in [0.4, 0.7]$, $U_T = 60\,\%$).

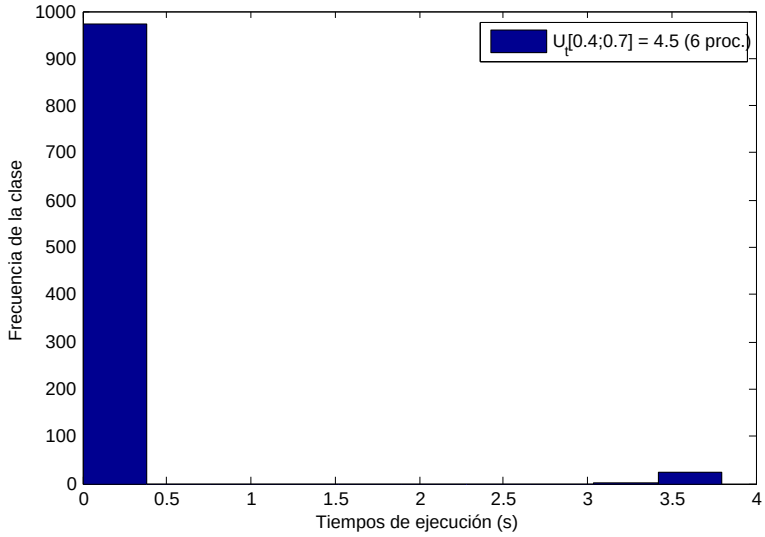


Figura A.20: Histograma de frecuencias de los tiempos de ejecución (6 procesadores, $u_i \in [0.4, 0.7]$, $U_T = 75\,\%$).

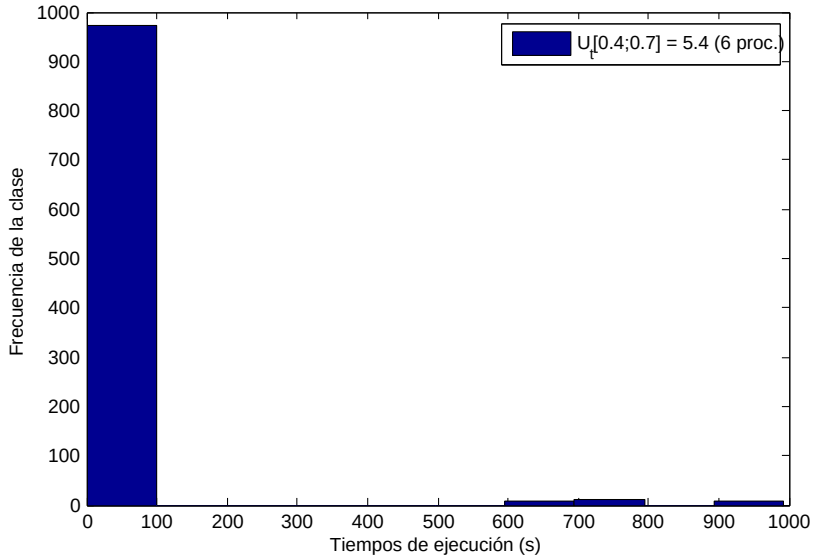


Figura A.21: Histograma de frecuencias de los tiempos de ejecución (6 procesadores, $u_i \in [0.4, 0.7]$, $U_T = 90\,\%$).

# Publicaciones del autor:

- Puente-Maury, L., Mejía-Alvarez, P., Leyva-del-Foyo, L. E. A Binary Integer Linear Programming-Based Approach for Solving the Allocation Problem in Multiprocessor Scheduling. $8^{th}$ *International Conference on Electrical Engineering, Computer Science and Automatic Control (CCE 2011)*. ISBN: 978-1-4577-1011-7.

# Bibliografía


[ABS95] Y. Oh A. Burchard, J. Liebeherr and S. H. Son. New strategies for assigning real-time tasks to multiprocessor systems. *IEEE Transactions on Computers*, 44(12):1429–1442, 1995.

[AHL60] A. G. Doig A. H. Land. An automatic method of solving discrete programming problems. *Econometrica*, 28(3):497–520, 1960.

[AJ00] B. Andersson and J. Johnsson. Fixed priority preemptive multiprocessor scheduling: to partition or not to partition. In *Proc. of the 7th IEEE International Conference on Real-Time Computing Systems and Applications*, pages 337–346, 2000.

[AS00a] J. Anderson and A. Srinivasan. Early-release fair scheduling. In *Proc. of the 12th Euromicro Conference on Real-Time Systems*, volume 12, pages 35–43, 2000.

[AS00b] J. Anderson and A. Srinivasan. Pfair scheduling: Beyond periodic task systems. In *Proc. of the 7th International Workshop on Real-Time Computing Systems and Applications*, 2000.

[AS01] J. Anderson and A. Srinivasan. Mixed pfair/erfair scheduling of asynchronous periodic tasks. In *Proc. of the 13th Euromicro Conference on Real-Time Systems*, 2001.

[AT06] B. Andersson and E. Tovar. Multiprocessor scheduling with few preemptions. In *Proc. of the 12th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, pages 322–334, 2006.

[BAJ03] S. Baruah B. Andersson and J. Johnsson. Static-Priority scheduling on multiprocessors. In *Proc. of the 22nd IEEE Real-Time Systems Symposium*, pages 193–202, 2003.

[Bak03a] T. P. Baker. An analysis of deadline-monotonic scheduling on a multiprocessor. Technical Report TR-030301, Florida State University, Department of Computer Science, 2003.

[Bak03b] T. P. Baker. Multiprocessor edf and deadline monotonic schedulability analysis. In *Proc. of the 24th IEEE Real-Time Systems Symposium*, pages 120–129, 2003.

[Bal65] E. Balas. An Additive Algorithm for Solving Linear Programs with Zero-One Variables. *Operations Research*, 13(3):517–549, 1965.

[Bar04] S. Baruah. Optimal utilization bounds for the fixed-priority scheduling of periodic task systems on identical multiprocessors. *IEEE Transactions on Computers*, 53(6):781–784, 2004.

[BB05] E. Bini and G. C. Butazzo. Measuring the performance of schedulability tests. *Real-Time Systems*, 3(1-2):129–154, 2005.

[BB08] S. Baruah and E. Bini. Partitioned scheduling of sporadic task systems: an ILP based approach. In *Proc. of the International Conference on Design and Architectures for Signal and Image Processing (DASIP 2008)*, 2008.

[BG03] S. Baruah and J. Goosens. Rate-monotonic scheduling on uniform multiprocessors. *IEEE Transactions on Computers*, 52(7):966–970, 2003.

[BK05] E. Burke and G. Kendall. *Search methodologies. Introductory Tutorials in Optimization and Decision Support Techniques*. Springer, 1st edition, 2005.

[CB11] B. Chattopadhay and S. Baruah. A lookup-table driven approach to partitioned scheduling. In *Proc. of the IEEE Real-Time Technology and Applications Symposium (RTAS)*, pages 257–265, 2011.

[DB09] R. Davis and A. Burns. A Survey of Hard Real-Time Scheduling Algorithms and Schedulability Analysis Techniques for Multiprocessor Systems. Technical Report YCS-2009-443, University of York, 2009.

[DL78] S. K. Dhall and C. L. Liu. On a real-time scheduling problem. *Operations Research*, 26(1):127–140, 1978.

[DM89] M. L. Dertouzos and A. K. Mok. Multiprocessor On-Line Scheduling of Hard-Real-Time Tasks. *IEEE Transactions on Software Engineering*, 15:1497–1506, 1989.

[DZM03] D. Mossé D. Zhu and R. G. Melhem. Multiple-Resource Periodic Scheduling Problem: how much fairness is necessary? In *Proc. of the Real-Time System Symposium*, pages 142–151, 2003.

[ECV98] S. Martello E.G. Coffman, G. Galambos and D. Vigo. *Bin Packing Approximation Algorithms: Combinatorial Analysis*. Kluwer Academic, 1998.

[Geo67] A. Geoffrion. Integer Programming by Implicit Enumeration and Balas Method. *SIAM Review*, 9(2):178–190, 1967.

[Geo69] A. Geoffrion. An Improved Implicit Enumeration Approach for Integer Programming. *Operations Research*, 17(3):437–454, 1969.

[GJ75] M. R. Garey and D. S. Johnson. Complexity Results for Multiprocessor Scheduling Under Resource Constraints. *SIAM Journal on Computing*, 4(4):397–411, 1975.

[Glo65] F. Glover. A Multiphase-Dual Algorithm for the Zero-One Integer Programming Problem. *Operations Research*, 13(6):879–919, 1965.

[Gom58] R. E. Gomory. Outline of an algorithm for integer solutions to linear programs. *Bulletin of the American Mathematical Society*, 64(5):275–278, 1958.

[HK04] D. Pisinger H. Kellerer, U. Pferschy. *Knapsack Problems*. Springer, 2004.

[HL05] F. S. Hillier and G. J. Lieberman. *Introduction to operations research*. Mc. Graw.Hill, 8th edition, 2005.

[IASAS08] F. Attia I. Al-Said and N. Al-Saiyd. Multiprocessor Scheduling Based on Genetic Algorithms. In *Accepted in The 2008 International Arab Conference on Information Technology (ACIT'2008)*, 2008.

[JCB04] P. Holman A. Srinivasan J. Anderson J. Carpenter, S. Funk and S. Baruah. A Categorization of Real-time Multiprocessor Scheduling Problems and Algorithms. In *Handbook on Scheduling Algorithms, Methods, and Models*. Chapman Hall/CRC, Boca, 2004.

[JMLG04a] J. L. Díaz J. M. López, M. García and D. F. García. Minimum and maximum utilization bounds for multiprocessor rate monotonic scheduling. *IEEE Transactions on Parallel and Distributed Systems*, 15(7):642–653, 2004.

[JMLG04b] J. L. Díaz J. M. López, M. García and D. F. García. Utilization bounds for edf scheduling on real-time multiprocessor systems. *Real-Time Systems*, 28:39–68, 2004.

[JPLD89] L. Sha J. P. Lehoczky and Y. Ding. The rate monotonic scheduling algorithm: exact characterization and average case behaviour. In *Proc. of the 10th IEEE Real-Time Systems Symposium*, pages 166–171, 1989.

[Kar84] N. Karmarkar. A New Polynomial-Time Algorithm for Linear Programming. *Combinatorica*, 4(4):373–395, 1984.

[KD08] B. Varghese A. Abraham F. Xhafa A. Daradoumis K. Dahal, A. Hossain. Scheduling in Multiprocessor System Using Genetic Algorithms. In *7th Computer Information Systems and Industrial Management Applications*, pages 281–286, 2008.

[KP95] B. Kalyanasundaram and K. Pruhs. Speed is as powerful as clairvoyance. In *Proc. of the Symposium on Foundations of Computer Science*, pages 214–221, 1995.

[Liu00] J. W. S. Liu. *Real-Time Systems*. Prentice Hall, 2000.

[LL73] C. L. Liu and W. Layland. Scheduling algorithms for multiprogramming in a hard real-time environment. *Journal of the ACM*, 20(1):46–61, 1973.

[Lun02] L. Lundberg. Analyzing fixed-priority global multiprocessor scheduling. In *Proc. of the 8th IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 145–153, 2002.

[LW82] J. Y. Leung and J. Whitehead. On the complexity of fixed-priority scheduling of periodic real-time tasks. *Performance Evaluation (Netherlands)*, 2(4):237–250, 1982.

[Meh92] S. Mehrotra. On the implementation of a primal-dual interior point method. *SIAM Journal on Optimization*, 2(4):575–601, 1992.

[NW06] J. Nocedal and S. Wright. *Numerical Optimization*. Springer, second edition, 2006.

[OB98] D. Oh and T. P. Baker. Utilization bounds for n-processor rate-monotonic scheduling with static priority assignment. *Journal of Real-Time Systems*, 15(2):183–192, 1998.

[OS95] Y. Oh and S.H. Son. Fixed priority scheduling of periodic tasks on multiprocessor systems. Technical Report CS-95-16, Univ. Of Virginia. Dept. of Computer Science, 1995.

[PEHJ08] A. M. Déplanche P. E. Hladik, H. Cambazard and N. Jussien. Solving a real-time allocation problem with constraint programming. *Journal of Systems and Software*, 81(1):132–149, 2008.

[Pet67] C. Petersen. Computational Experience with Variants of the Balas Algorithm Applied to the Selection of R&D Projects. *Management Science*, 13(9):736–750, 1967.

[POY07] J. He P. Oliveto and X. Yao. Time Complexity of Evolutionary Algorithms for Combinatorial Optimization: A Decade of Results. *International Journal of Automation and Computing*, 3(4):281–293, 2007.

[PZMA04] O. Pereira Zapata and P. Mejía Alvarez. Edf and rm multiprocessor scheduling algorithms: Survey and performance evaluation. Technical Report CINVESTAV-CS-RTG-02, CINVESTAV-IPN, Computer Science Department, 2004.

[Rao96] S. S. Rao. *Engineering Optimization: Theory and Practice*. Wiley, third edition, 1996.

[RR07] S. Rajasekaran and J. Reif. *Handbook of Parallel Computing: Models, Algorithms and Applications*. Chapman & Hall/Crc Computer & Information Science Series, 1st edition, 2007.

[SB02] A. Srinivasan and S. Baruah. Deadline-based scheduling of periodic task systems on multiprocessors. *Information Processing Letters*, 84(2):93–98, 2002.

[SBV96] C. G. Plaxton S. Baruah, N. Cohen and D. Varvel. Proportionate progress: a notion of fairness in resource allocation. *Algorithmica*, 15:600–625, 1996.

[SFB03] J. Goosens S. Funk and S. Baruah. On-line scheduling on uniform multiprocessors. *Real-Time Systems*, 25:197–205, 2003.

[SKBP95] J. Gehrke S. K. Baruah and C. G. Plaxton. Fast scheduling of periodic tasks on multiple resources. In *Proc. of the 9th International Parallel Processing Symposium*, pages 280–288, 1995.

[SLM03] R. Melhem S. Lauzac and D. Mosse. An improved rate-monotonic admission control and its applications. *IEEE Transactions on Computers*, 52(2):337–350, 2003.

[Tai93] E. Taillard. Benchmarks for basic scheduling problems. *European Journal of Operational Research*, 64(2):278–285, 1993.

[Wri97] S. J. Wright. *Primal-Dual interior-point methods*. SIAM, 1997.