# Modelos y Métodos de Investigación de Operaciones. Procedimientos para Pensar.

## (Modelado y Resolución de Problemas de Organización Industrial mediante Programación Matemática)

**Nota: Este documento está sujeto a cambios. Se agradecerá cualquier comentario que pueda ayudar a mejorarlo. Enviar un correo a jpgarcia@omp.upv.es indicando el número de página y el número de línea con la sugerencia.**

Métodos Cuantitativos de Organización Industrial

*José Pedro García Sabater, Julien Maheut*

*Grupo de Investigación ROGLE*

*Departamento de Organización de Empresas*

*Curso 2011 / 2012*

# 1 MODELOS

## 1.1 ¿Qué es un Modelo?

Según una de las definiciones más simples de modelo es la propuesta por Colin Lee (1972)

*"Un modelo es una representación de la realidad"*

Esta definición es llamativa por su simplicidad pero no aclara el porqué de los modelos.

Pidd (1996) propone la siguiente definición mucho más completa:

> *"Un modelo es una representación explícita y externa de parte de la realidad como la ven las personas que desean usar el modelo para entender, cambiar, gestionar y controlar dicha parte de la realidad"*

De esta definición se pueden extraer muchas reflexiones interesantes sobre los Modelos y su uso en Investigación de Operaciones. Quizá la más relevante es que los modelos son representaciones (no son la realidad, que, por cierto, se asume que existe).

En palabras de George Box (1987)

*"Básicamente todos los modelos son erróneos, aunque algunos son útiles"*

Desgraciadamente el ser humano tiende a confundir el modelo con la realidad. En un proceso de metonimia el ser humano tiende a crear modelos y tergiversar la realidad hasta que esta se adapta a ellos. Pero eso es un defecto de la mente humana, no del proceso de modelar.

Los modelos son explícitos se construyen manejan y modifican como tales. Y aunque no se debe confundir el modelo con la realidad, el modelo debe tener una imagen física sobre la que los diferentes actores puedan opinar.

Los modelos son externos. Mientras no tienen una representación externa respecto del modelador son simplemente una teoría mental del mismo. En esa presentación externa radica una de las grandes ventajas de los modelos: ponen negro sobre blanco los pensamientos, los datos, las hipótesis y las intuiciones. Este esfuerzo es más que suficiente, en algunas ocasiones, para reconocer que no hay tal modelo sino un conjunto de prejuicios. En la mayor parte de las ocasiones la mera representación explícita y externa de partes del modelo, permite a los actores del proceso tomar decisiones en beneficio del sistema que alteran sustancialmente el propio sistema modelado.

Los modelos representan parte de la realidad. Afortunadamente la realidad es siempre más compleja que cualquier modelo por sofisticado que éste sea. El modelador discrimina qué aspectos son relevantes y cuáles no, en función del objetivo que pretende alcanzar.

Los modelos los realizan los modeladores. Son su esfuerzo y son su resultado. En muchas ocasiones, al construir modelos, el modelador ha de atender comentarios que le obligan a incorporar uno u otro aspecto de la realidad. Entonces el modelador es otro. Si a un pintor le dijeran el color con el que ha de pintar una puesta de sol, no sería su puesta de sol sino la del observador impertinente. En este caso la

realidad representada se limita a ser la que quiere ver, manejar controlar o cambiar el que dirige el modelo. En muchas ocasiones una empresa solicita un modelo a un consultor externo para poderlo utilizar internamente. Es uno de los caminos más adecuados para que el modelo no se pueda utilizar, pues siempre habrá matices que podrían haber sido representados de otra manera y que pueden entrar en conflicto con las expectativas de lo que tiene que hacer el modelo. Pero por otro lado la visión de un espectador externo permite plantear e incluso resolver conflictos que subyacían implícitamente en la versión que cada actor tenía de los hechos.

Los modelos, al representar externa y explícitamente parte de la realidad, permiten fundamentalmente entender. Una etapa bastante habitual en el ciclo de vida de un modelo exige, tras uno (o varios) intentos de modelado, cambiar de herramienta de modelado, puesto que el mejor entendimiento del problema provoca cambios radicales en la percepción de la realidad.

La inteligencia de la realidad a través del modelo, permitirá asesorar sobre la oportunidad de cambios en la realidad modelada. Dichos cambios serán más adecuados cuantos más aspectos de la realidad se hayan podido modelar.

La gestión de la realidad a través de los modelos es una herramienta de continua, pero no percibida, aplicación de los mismos. La gestión financiera de una empresa es uno de los modos más extendidos de uso de un modelo (la contabilidad) sin una percepción clara del modelo utilizado. Pero también el uso de las técnicas *Material Requirement Planning* (MRP) en la gestión de materiales responden a la implementación informática de modelos en función de los cuales se toman decisiones. Incluso la implementación informática es en ocasiones inadecuada porque no se conocía suficientemente bien la realidad (o lo que es peor, el modelo que lo representaba).

Por último, aunque probablemente uno de los aspectos más relevantes: el modelo, al anticipar resultados, permite establecer referencias en función de las cuales medir o controlar el rendimiento de un sistema. Con las evidentes implicaciones que esto tiene en el control y mejora de los procesos.

## *1.2 ¿Para qué sirve un modelo?*

En atención a lo anterior se pueden definir tres ámbitos de utilidad de los modelos en la Investigación de Operaciones:

- Aprender / Entender
- Implementar en un ordenador
- Tomar decisiones

### 1.2.1 Aprender / Entender

En primer lugar hay que destacar que la experiencia demuestra que el principal beneficio en la generación de un modelo es el entendimiento que el modelador adquiere del comportamiento de la realidad. Puede ocurrir, y de hecho ocurre con frecuencia, que una vez finalizado el modelo, los

objetivos perseguidos inicialmente se hayan alcanzado sin hacer ningún tipo de experimento. Modelar como proceso puede tener más valor que el modelo en sí mismo

Es habitual que para desarrollar un modelo se tenga que acceder a información a la que nunca se le habría prestado atención. Asimismo es común que la generación de modelo haga aparecer datos "reales y contradictorios" entre diferentes elementos de la realidad.

Una vez construido el modelo, se puede utilizar su ejecución para conocer como el sistema actúa y reacciona. Es, por ejemplo, el caso de los "simuladores de vuelo" utilizados para el entrenamiento de los futuros pilotos.

Además el modelo, como representación externa y explícita, puede permitirnos conocer errores y fundamentalmente (de)mostrarlos. De tal modo que el responsable del error pueda reconocer sus equivocaciones sin que nadie tenga que "decírselo a la cara" (lo dice el ordenador).

Por último, dentro de este epígrafe, podemos destacar la utilidad de los modelos como base de discusión. Si el modelo representa la realidad, los gestores de ésta podrán probar las ventajas de sus opiniones sobre el modelo, centrando de este modo la discusión hacia aspectos realizables y rentables.

### 1.2.2 Implementar en un ordenador

La automatización de procesos exige el modelado previo. Si se desea gestionar la información que genera una empresa, o implementar un sistema de gestión de recursos humanos es necesario realizar un modelo de dicha empresa que comprenda de la manera más eficiente posible toda la información vinculada. Cuanto más general sea el modelo, mayor será la cantidad de empresas a las que se las podrá aplicar el mismo *software*.

Del mismo modo la utilidad de los modelos de Programación de Producción viene justificada, en gran medida, por la capacidad de éstos de ser implementados y resueltos mediante sistemas informáticos que puedan automatizar el proceso de toma de decisión.

### 1.2.3 Toma de decisiones

Los modelos construidos permiten mediante su resolución ayudar a la toma de decisiones generando soluciones óptimas dado un objetivo establecido.

Asimismo pueden ser utilizados para evaluar el impacto de tomar decisiones, antes de tomarlos, y de este modo elegir la que más se ajuste a la solución.

Pero además, desarrollar el modelo, ejecutarlo y analizar las soluciones permite objetivar el proceso de análisis, permite "pintar una realidad" que todos tienen que aceptar, o aportar datos que mejoren el modelo.De este modo, al objetivar el proceso de análisis, los participantes en el proceso de toma de decisiones entran en una dinámica de objetivación y aporte de datos, que simplifica y favorece el propio proceso y su resultado.

## 1.3 El Cliente y el Problema

En general el cliente no conoce suficientemente bien su problema. Si lo conociera no nos solicitaría ayuda para resolverlo, porque en muchas ocasiones ser capaz de explicitar el problema es hacer evidente una solución.

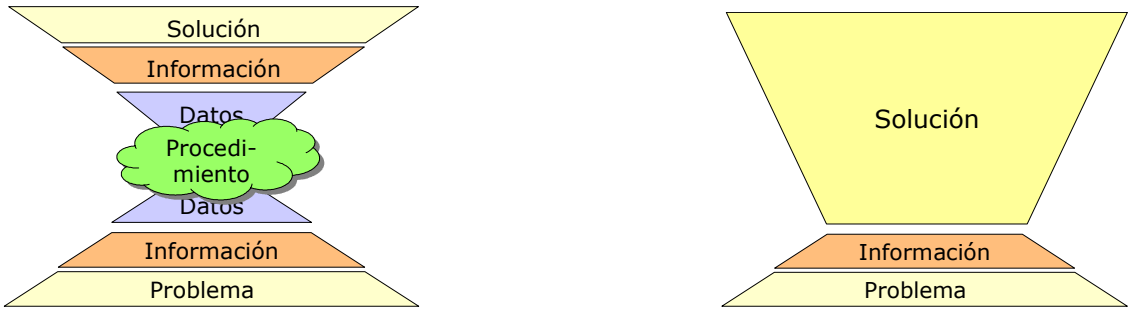
En general se puede afirmar que el cliente tiene una "nebulosa" sobre un problema a la que damos una cierta forma con la construcción del modelo. Es por ello que muchos problemas de organización industrial *no se resuelven sino que se disuelven*.

Pero además el número de actores implicados en un problema es en general alto y con perfiles muy diferenciados entre sí. En la mayor parte de las ocasiones el conocimiento sobre el problema real, sobre las circunstancias del mismo es pobre. Escasísimo es el conocimiento sobre las herramientas para resolver dicho problema. Y ese desconocimiento sobre la herramienta es un factor a tener en cuenta porque permite "vender" la solución, pero impide la "compra".

Sin ánimo de ser exhaustivo se pueden identificar al menos 4 tipos de implicados en la generación de una solución. Quizá el principal de los actores es el **pagano**: el que paga el análisis, desarrollo y puesta en marcha de la solución. Las motivaciones del pagano son diferentes a las del **usuario**, la persona cuya "vida se verá mejorada" por la solución y resolución del problema. Toda aproximación tiene siempre uno o varios **sufridores.** Los sufridores verán afectada su vida por la modificación de su "entorno" y es imposible para alguien que no sea ellos mismos (e incluso para ellos mismos) saber hasta que punto su calidad de vida empeorará por la resolución del problema. Por último existe, generalmente, una tipología de cliente que recibe el nombre de **prescriptor.** En principio este personaje (básico en el funcionamiento de la economía moderna) no gana nada con una resolución adecuada del problema, pero pierde en el caso de que (como esperan los sufridores) la solución no sea finalmente adoptada.

## 1.4 El Problema, y el Concepto de Solución

A partir de la descripción del problema, generamos información que podrá ser capturada en forma de datos, dichos datos son analizables de tal modo que generan unos datos conocidos como variables de salida. Esos datos por lo general son inútiles salvo que se transformen en información, y sólo ésta última puede ayudar a resolver el problema.




Así pues existen tres significados para la palabra solución en nuestro entorno. Una solución es un conjunto de variables que han adquirido un determinado valor. Una solución es también el programa que

generará información a partir de los datos disponibles (generalmente se le denomina solución informática) Y por último está lo que el cliente considera que es la solución a su problema, que básicamente se da cuando el problema desaparece.

## *1.5 Ciclo de Vida de la construcción de Modelos*

No existe un método para construir un modelo perfecto de modo directo. En cualquier caso se puede decir que en la definición de cualquier modelo hay tres etapas o hitos básicos que se concretan en:

1. Definir el Problema. Esta fase incluye entender el problema y acordar con el cliente los resultados a obtener.

2. Modelar y Construir la Solución. Esta fase incluye definir el tipo de técnica a utilizar, generar el modelo (implementarlo informáticamente si es el caso) y por último validarlo.

3. Utilizar la Solución. Un modelo perfecto que no se utilice es un modelo perfectamente inútil. Ser capaz de implementar el modelo de tal manera que el cliente lo utilice, y mantener un concreto sistema de actualización son los dos elementos básicos de esta fase.

Cualquiera de las etapas citadas exige replantearse siempre la vuelta al principio del proceso. La mejor comprensión de la realidad puede llevar (lleva) a cambiar el tipo (o tipos) de técnica a utilizar para alcanzar el objetivo propuesto.

### 1.5.1 Definir el Problema

1. Entender el problema: Hay que estructurar el problema para entenderlo. Cualquier herramienta es buena. En ocasiones con esta etapa el problema a resolver queda resuelto. Y en general también ocurre que el primer problema planteado no era el problema real.

2. Acordar con el cliente los resultados a obtener. No significa necesariamente que el cliente deba definir el resultado concreto del trabajo. Pero es interesante conocer si pretende una respuesta del tipo sí o no o una hoja Excel.

### 1.5.2 Modelar y Construir la Solución

3. Definir el tipo de técnica. La decisión del tipo de técnica que mejor se ajusta al problema puede ser revocada en cualquier instante, pero da por perdido todo el trabajo anterior. Esto incluye el análisis de datos disponibles y resultados requeridos.

4. Generar el modelo. Esta etapa incluye estimar los parámetros para modelar o calcular resultados, además de dar forma física al modelo. En este punto es de destacar la aplicación del principio “Ir paso a paso”. Esto implica abordar escalonadamente los diferentes aspectos de la realidad que se pretenden modelar.

5. Validar el modelo. Decidir si el modelo vale para algo, si se puede usar y si el cliente lo encontrará aceptable. Fundamentalmente esta fase exige comprobar que se comporta como se pretendía que se comportara.

### 1.5.3 Utilizar la Solución

6. Implementar el modelo. Trabajar con el cliente para poder extraer los máximos beneficios del trabajo realizado.

7. Actualizar el modelo. Es evidente que si la realidad es cambiante el modelo debe adaptarse a las nuevas circunstancias de manera continua si se pretende que siga teniendo utilidad.

## *1.6 Generación de Soluciones Informáticas para la resolución de Problemas que se abordan mediante métodos matemáticos*

A continuación se ofrece una estructura útil para ayudar a plantear herramientas que resuelven problemas reales de clientes reales.

1) Análisis del Problema. En general el cliente no es capaz de definir su problema. A veces, pensará que lo puede definir pero seguramente a lo largo del proceso se redefinirá el problema en varias ocasiones.
2) Preproceso de Datos. Analizar los datos disponibles nos permite tener una visión detallada del problema. En general, si los datos no han sido previamente utilizados serán probablemente erróneos. Este paso, igual que los siguientes, puede suponer analizar de nuevo el problema.
3) Generar resultados a mano. Si somos capaces de entender el problema seguro que somos capaces de generar una hoja de cálculo donde representar una posible solución, calculada a mano.
    a. Definir la Estructura de la Solución
    b. Diseñar e implementar un Representador de una Solución
    c. Diseño del Método de Evaluación de Resultados (definir cómo se va a presentar que un resultado es mejor que otro)
    d. Implementar el Evaluador de Solución
4) Comprobar que los resultados son inteligibles por parte del cliente. Y que lo que creemos que es peor es realmente peor.
5) Generar un modelo de programación matemática (preferiblemente lineal).
6) Selección de la Herramienta o Tecnología de resolución adecuada.
7) Definir el Procedimiento de Resolución

8) Diseño de Procedimientos de Resolución para clientes no-expertos.
    a. Diseño de Estructura de Datos de Entrada
    b. Definir los Procedimientos de Lectura y captura de Datos
        - Desde bases de datos estructuradas existentes y/o nuevas
        - Desde ficheros de texto
        - Desde pantalla
    c. Generar Algoritmos para generar soluciones
        - Generación aleatoria de Soluciones
        - Heurísticas Voraces
        - Diseño de Métodos de Enumeración
        - Diseño de Meta-heurísticas

## *1.7 Algunos principios para tener éxito en el modelado*

Aunque como se verá posteriormente existen múltiples tipos de modelos (y por tanto de procesos de modelado) se pueden extraer algunos principios generales útiles en cualquier caso:

- Los Modelos han de ser simples, su análisis debe ser complejo
- Ir paso a paso
- Usar metáforas, analogías y similitudes
- Los datos disponibles no deben conformar el modelo.
- Modelar es explorar.

### 1.7.1 Los Modelos han de ser simples, su análisis debe ser complejo

Al modelar se puede tener la tendencia de trasladar toda la complejidad de la realidad al modelo. Esto, aunque suele agradar al que "mira" el modelo, no es útil para quien lo debe utilizar por dos motivos: un modelo de este tipo es difícil de construir y también es difícil de utilizar.

Antes de comenzar el proceso de modelado se debería responder a la pregunta: "¿Para qué quiero y necesito el modelo?" de un modo concreto. De este modo se puede garantizar que para hacer un simulador de coches, no se pierde tiempo modelando el funcionamiento del turbo cuando lo que se pretende es hacer una herramienta para comprender el funcionamiento del volante.

### 1.7.2 Ir paso a paso

Es habitual observar que se pretende construir un modelo considerando todos los aspectos simultáneamente. La ciencia avanza paso a paso, los modelos, si pretenden estar dentro de ella, también.

Metafóricamente hablando, intentar construir un modelo completo desde el principio puede llevar a que, al intentar dibujar las hojas en los árboles, se olvide que el objeto a pintar era el bosque.

Un corolario de este principio exige “Dividir para Vencer”. Empezar generando pequeños modelos de una parte reducida y determinada del proceso para aumentar su *scope* gradualmente.

De este principio se deriva que estrategia recomendable es evitar tener todos los aspectos en cuenta desde el principio. El proceso de modelado puede comenzar aislando una pequeña parte y realizando un modelo detallado, que permita su reproducción en otras secciones. También se puede realizar al principio un modelo muy general, e ir mejorando etapa a etapa la exactitud del mismo.

### 1.7.3 Usar al máximo metáforas, analogías y similitudes

Un buen modelador, más que quedar re*string*ido por la realidad tal y como se percibe a primera vista, es interesante abordarla, e incluso modelarla desde otros puntos de vista.

El punto anterior se pretenden ilustrar con este ejemplo: Si al hacer un modelo de secuenciación para líneas de montaje en sistemas Justo-a-Tiempo se cae en la cuenta de que el problema matemático es básicamente el problema de repartir escaños de una manera proporcional, se habrá conseguido resolver un problema actual (secuenciación JIT) con herramientas desarrolladas desde el siglo XVIII. Todas las propiedades analizadas para el segundo problema, pueden ser adaptadas para el primero más nuevo.

Además, el abandonar de modo explícito la realidad puede simplificar el problema o representarlo de un modo más sencillo. Cualquier plano de metro de cualquier ciudad no representa cada línea y cada estación tal cual es sino que une mediante líneas, los puntos, que representan estaciones, que en casi ningún caso pueden superponerse sobre un plano detallado y proporcional de la ciudad. La representación exacta de la realidad incrementaría la dificultad en la lectura de dichos planos.

### 1.7.4 Los datos disponibles no deben conformar el modelo

Un fallo común a la hora de plantear un modelo es retrasar el comienzo del modelado hasta que se disponga de los datos. El planteamiento debe ser el contrario, el modelo debe requerir datos, no los datos conformar el modelo.

El analista debe desarrollar las líneas básicas sobre el modelo y una vez hecho esto, debiera definirse la estructura de datos necesarios. Si hubiera tiempo lo lógico sería que a la luz de estos resultados preliminares se redefiniera el modelo y por tanto los datos necesarios, y así sucesivamente.

Se pueden distinguir tres conjuntos básicos de datos necesarios para crear y validar un modelo:

- Datos que aportan información preliminar y contextual. Permitirán generar el modelo.
- Datos que se recogen para definir el modelo. Estos datos nos permitirán parametrizar el modelo.
- Datos que permiten evaluar la bondad del modelo.

Hay que destacar la importancia de que los datos del segundo y el tercer tipo sean distintos, porque en caso contrario el modelo no se habrá realmente validado.

Una última recomendación respecto a los datos es evitar aquellos que ya están recogidos. Sirva el siguiente clásico ejemplo como ilustración.

Para desarrollar un modelo para la programación de producción puede ser necesario desarrollar submodelos de demandas de los clientes para los productos fabricados. La mayor parte de las empresas guardan esta información en sistemas.

El modo obvio de recoger la información de la demanda es acudir a los sistemas informáticos que se usan para introducir órdenes y enviar facturas. Pero no es conveniente tomar ese camino tan evidente. Los sistemas sólo recogen lo que se vende, no lo que el cliente quiere. Por tanto las ventas muchas veces simplemente reflejan lo que hay en stock, pues se obliga al cliente a comprar lo que existe."

### 1.7.5 Principio Subyacente: Modelar es Explorar

Dado que un modelo es el resultado de intentar representar parte de la realidad para tomar decisiones, implementar o entender, se podría pensar que el proceso de modelar es un proceso lineal.

Sin embargo, la experiencia muestra que en el proceso de modelar hay muchas vueltas atrás, cambios de dirección o cambios de perspectiva, incluso es bastante habitual que haya cambios de herramientas.

Modelar es explorar la realidad. Y en especial la realidad desconocida. Por ello siempre aplica el siguiente corolario de la ley de Murphy: "*Si se consigue que el modelo funcione a la primera, es que se ha errado el problema"*.

# 2 TIPOS DE MODELOS. LOS MODELOS DE PROGRAMACIÓN MATEMÁTICA

## 2.1 Clasificación de modelos según el efecto de su resolución

*Shapiro (2001)* clasifica los modelos según el efecto de su resultado en Normativos o Descriptivos.

Son normativos los modelos matemáticos (a su vez estos se pueden clasificar en modelos de optimización y modelos de resolución mediante heurísticas).

Los modelos descriptivos que engloban al resto (Previsión, *Data Mining*, Simulación, Dinámica de Sistemas,…).

### 2.1.1 Modelos Normativos

Los modelos normativos exigen el planteamiento de un modelo matemático (probablemente en forma de función objetivo y restricciones). Los modelos cuya estructura se ajusta a algunos de los patrones clásicos para los que es factible la optimización (programación lineal por ejemplo) forman el subconjunto de modelos de optimización.

En ocasiones la estructura del modelo impide el uso de algún método de optimización conocido, es por ello que se plantean los procedimientos heurísticos de resolución que, si bien no garantizan óptimos, permiten encontrar soluciones en espacios cortos de tiempo.

Es evidente que el trabajo en el primer caso se debe centrar en el proceso de modelado, mientras que en el segundo grupo el esfuerzo se hace en la definición del método heurístico de resolución.

En este libro se despliega uno de los tipos de modelos normativos, la Programación Matemática, y más concretamente la Programación Lineal Entera. La Programación Matemática no es el único modo de modelar matemáticamente, ni el único modo normativo de hacerlo. Por ello en los puntos siguientes se hará una presentación de algunos de estos modos.

### 2.1.2 Modelos Descriptivos

Los modelos descriptivos abarcan todas aquellas técnicas de modelado que no comportan la definición de estructuras matemáticas que definen una solución como la deseable para ser implementada.

Entre los modelos descriptivos se pueden citar los modelos de simulación, la teoría de colas e incluso las técnicas de previsión entre otras. Algunos de los modelos descriptivos llevan aparejada una carga matemática importante, mientras que otros su estructura no es de tipo matemático. Aunque ello no les quita ni un ápice de formalidad. Por poner un ejemplo los modelos IDEF-0 son altamente formales y estándar. Aunque tienen aspecto de grafo, no necesariamente debieran ser incluidos entre los que se denominan Modelos Matemáticos.

## 2.2 Modelos Matemáticos

Los modelos matemáticos son modelos formales que utilizan el lenguaje de las matemáticas para describir un sistema, expresando parámetros, variables, relaciones. El lenguaje matemático no se limita a la expresión de números y operadores aritméticos que los relacionan. Así por ejemplo la teoría de grafos, ampliamente utilizada en aplicaciones prácticas, es un "*subconjunto*" de la más general teoría de conjuntos.

Los modelos matemáticos se pueden clasificar de múltiples maneras. A continuación se describen algunas que se consideran relevantes.

Los modelos pueden ser estáticos o dinámicos. Un modelo estático no tiene en cuenta el tiempo, mientras que los modelos dinámicos sí. Los modelos dinámicos se suelen representar con ecuaciones en diferencias o ecuaciones diferenciales.

Los modelos pueden ser lineales o no-lineales. Si todos los operadores de un modelo son lineales el modelo es lineal, si al menos uno es no-lineal el modelo es no-lineal. Aunque hay excepciones, los modelos lineales son mucho más fáciles de manejar que los modelos no-lineales. En general los modelos no-lineales pueden ser linealizados, pero entonces, es posible, que se estén perdiendo aspectos relevantes del problema.

Un modelo puede ser determinista o estocástico. Un modelo determinista es aquel en que cada conjunto de variables en un estado está definido por los parámetros del modelo y por los estados anteriores. Un modelo determinista se comporta siempre igual para un conjunto de parámetros de entrada. En un modelo estocástico las variables de estado se representan por distribuciones de probabilidad, y por tanto el modelo es capaz de recoger aleatoriedad o incertidumbre.

## 2.3 Modelos de Programación Matemática

La característica común que comparten todos los modos de modelar matemáticamente es que representan la realidad mediante variables y parámetros (y algunos otros *artificios* como funciones o conjuntos). De este modo la realidad queda cuantificada. Entre ellos están la Programación Dinámica o la Teoría de Grafos.

Los modelos de Programación Matemática se distinguen por que representan la realidad mediante funciones. Estas son combinación de variables y parámetros en forma de restricciones y/o funciones objetivo. En general, las restricciones se deben respetar y las funciones objetivo establecen la diferencia entre una solución y otra mejor..

Este tipo de modelos matemáticos pertenecen al grupo de los modelos normativos (qué indican el camino a seguir) frente a la categoría de los descriptivos (que describen la situación actual o futura).

### 2.3.1 ¿Por qué se llama Programación Matemática?[2]

Tres nombres de tres científicos ilustres van asociados al origen del extraño nombre de "Programación Matemática": Koopmans, Dantzig y Kantorovich[3].

Los tres parecen haber diseñado métodos de Planificación y Programación de Operaciones (producción y transporte) utilizando modelos matemáticos.

---

[2] Fue el profesor Companys de la Universidad Politécnica de Catalunya quien me puso detrás de la pista de esta interesante historia. A él le debe uno de los autores de este libro muchas cosas interesantes que he aprendido, y esta es sólo una de ellas.

[3] Al finalizar el primer tercio del siglo XX, Kantorovich, premio Nobel de Economía en 1975, se enfrenta al problema de planificación desde una óptica de Optimización Matemática. Kantorovich, que vivía en la Unión Soviética enfoca cómo combinar la capacidad productiva de una fábrica para maximizar la producción. Para ello utiliza un método de análisis que posteriormente se llamó Programación Lineal. Aunque entonces no tenía nombre.

En el año 1951 Koopmans (que fue premiado junto con Kantorovich con el Nobel) edita un libro de título *"Activity Analysis of Production and Allocation*", (Wiley, New York (1951)). Dicho libro recoge trabajos que sus autores dicen que son ampliaciones o reducción de trabajos publicados entre 1947 y 1949.

En un libro que él mismo edita, Koopmans, escribe dos capítulos relevantes. El capítulo III "Analysis of production as an efficient combination of activities" donde expone un "problema de producción" de manera matemática, y el capítulo XIV de título "A model of transportation" donde planteó el problema de "programar el transporte de barcos" también desde una óptica de optimización matemática. Unos años antes había planteado el problema pero de modo teórico según él mismo indica.

De hecho en los artículos indica que estaba influenciado por una corta entrevista que tuvo con Dantzig. El propio Dantzig escribe el capítulo II del citado libro de título: "The programming of interdependent activities: Mathematical Model" que indica que es una revisión de un artículo de 1949. En ese capítulo se distingue la palabra "programming" que hace referencia a la programación y "Mathematical" que hace referencia al modelo. Dantzig se concentra en los modelos donde las relaciones son lineales pues tienen unas propiedades interesantes, entre otras que no hay óptimos locales, y de repente en la página 30 los problemas de programación con modelos lineales se convierten en "problemas de programación lineal". Aparentemente el nombre de Programación Lineal fue sugerido por Koopmans en 1948 en una reunión que tuvieron Koopmans y Dantzig en RAND Corporation.

La nota de entrada del capítulo nos recuerda que está "republicando un trabajo de 1949". Porque ya en 1947 Dantzig había diseñado el algoritmo del Simplex, que es un procedimiento eficaz de resolución del problema de programación lineal.

Según una historia paralela, el término programación lineal habría surgido porque "programación" era a lo que se dedicaba el departamento en la USAF para el que trabajaba Dantzig. El propio Dantzig, sugiere que inicialmente su método se utilizó para calcular las dietas óptimas.

¿Y Kantorovich? En su autobiografía para el Premio Nobel, Kantorovich escribe: "In 1939, the Leningrad University Press printed my booklet called *The Mathematical Method of Production Planning and Organization* which was devoted to the formulation of the basic economic problems, their mathematical form, a sketch of the solution method, and the first discussion of its economic sense. In essence, it contained the main ideas of the theories and algorithms of linear programming. The work remained unknown for many years to Western scholars. Later, Tjalling Koopmans, George Dantzing, et al, found these results and, moreover, in their own way. But their contributions remained unknown to me until the middle of the 50s."

Se podría decir pues que se lo que se conoce como Programación Matemática fue originariamente un modo de resolver problemas de Programación mediante métodos matemáticos.

Los modelos de programación cuyas variables tenían relaciones lineales, tenían la interesante propiedad de no tener óptimos locales… por lo que la Programación Lineal se convirtió pronto en un lugar común de encuentro de *modeladores y solucionadores.*

### 2.3.2 Una clasificación de Modelos de Programación Matemática

Una clasificación de los modelos de programación matemática podría tener en cuenta las siguientes características:

- Estructura, objetivos y restricciones (lineales o no-lineales)
- Características de las Variables (Reales, Discretas -Enteras-, Binarias)
- Certidumbre de los Parámetros (Ciertos e Inciertos)
- Número de Objetivos (Ninguno, Uno o más de Uno)
- Número de Restricciones (Ninguna, Más de Cero)

El objeto de esta descripción no es establecer una perfecta clasificación de todos los modelos de programación matemática. En realidad se pretende fijar un marco que sirva de referencia en el contexto de estos apuntes.

#### *2.3.2.1 Programación Lineal*

Entre los tipos de modelos de uso más generalizado en Programación Matemática se encuentra la denominada Programación Lineal. Ésta, en su forma más básica, consiste en un conjunto de variables reales, que mediante combinación lineal de parámetros ciertos, permite establecer un objetivo y restricciones lineales.

Los fundamentos matemáticos de los modelos lineales se encuentran en la teoría de las desigualdades lineales desarrollada en el siglo XIX como se puede leer en (Poler 2001). Aunque se encuentran precedentes en distintos campos (teoría de juegos, definición de dietas, problemas de transporte...), la formulación y resolución general de los problemas de Programación Lineal fue realizada en el proyecto SCOOP, lanzado en 1947 por el ejército del aire de los Estados Unidos de Norteamérica, dando lugar al algoritmo denominado Simplex expuesto inicialmente por Dantzig en 1947. En menos de 10 años la Programación Lineal experimento un fuerte desarrollo con trabajos que abordaron, entre otros temas, la degeneración, la dualidad y las formas compactas.

Actualmente es posible encontrar en el mercado, incluyendo aplicaciones gratuitas en internet, aplicaciones comerciales para la resolución eficiente de problemas de Programación Lineal (GUROBI, CPLEX, XPRESS, LINDO, QSB...), siendo un avance significativo de los últimos años el desarrollo de paquetes que facilitan la introducción del modelo y la integración de éste con los Sistemas de Información de la empresa.

La mayor parte de estos paquetes utilizan (o han utilizado) el denominado método Simplex. Dicho método, aunque computacionalmente ineficiente, tiene la ventaja docente de ser metódico y que permite explicar, mediante el propio método, algunos conceptos como precios-sombra o costes reducidos.

Hasta finales de la década de los 80 del siglo XX no surgen como alternativa válida los denominados métodos del punto interior. El menor coste computacional de este tipo de algoritmos hace que su implantación en los paquetes comerciales sea creciente.

Por último parece necesario destacar que aunque para el observador no experimentado la exigencia de linealidad puede parecer excesivamente restrictiva, la realidad es que un gran número de problemas reales puedan ser modelados con esa consideración (Williams, 1999). La ventaja de los Programación Lineal frente a la Programación No-Lineal es que para esta no se conocen modelos generales de resolución eficientes. Curiosamente el trabajo que se desarrolla en resolución de Programación No-Lineal está colaborando en mejorar la eficiencia de la Programación Lineal.

#### 2.3.2.2 *Programación Lineal Entera*

Si a alguna de las variables de un problema lineal se le impone la condición de integridad el problema pasa a ser de Programación Lineal Entera Mixta. Si todas son variables enteras, el problema pasa a ser de Programación Lineal Entera.. La condición de integridad puede venir impuesta, entre otros motivos, por el imposible fraccionamiento de determinados recursos. Uno de los procedimientos más efectivos para la resolución de este tipo de problemas se fundamenta en el concepto de ramificación y cota. Desgraciadamente aunque la lógica de este procedimiento es eficaz, conduciendo necesariamente al óptimo, el coste computacional en algunos problemas es, aún hoy en día, excesivo. Otro procedimiento para la resolución de estos problemas se basa en los métodos de planos cortantes, aunque este método levantó grandes expectativas por ahora no han fructificado de modo eficiente.

Una variante especial de los problemas de Programación Lineal Entera lo constituyen aquellos donde algunas variables son bivalentes. El uso de este tipo de variables tiene su origen en la representación de aquellas decisiones que sólo admiten dos valores, pero también aquellos problemas que exigen restricciones de tipo lógico. La pretensión de resolver estos problemas de modo eficiente ha dado lugar a métodos como el de enumeración implícita o técnicas más generales de enumeración como las descritas en (Kauffmann y Henry-Labordere, 1976).

Hay que destacar la existencia de algunos tipos especiales de problemas con variables bivalentes, que se abordan mediante métodos específicos de resolución, óptimos en algunos casos y más eficientes por haber sido desarrollados “ex profeso”. Algunos de estos problemas son los de cubrimiento, asignación, partición, mochila y rutas (Williams, 1999).

#### *2.3.2.3 Programación Estocástica*

Si a los problemas de Programación Matemática (en general) se les incorpora la incertidumbre en los parámetros, esta incertidumbre se puede abordar mediante la denominada Programación Estocástica.

Una variante de la misma especialmente interesante es la Programación Lineal Estocástica, que puede ser resuelta de modo óptimo, aunque con un coste computacional elevado.

Uno de los mecanismos para abordar la incertidumbre en los datos es el uso de los denominados escenarios. Estos constituyen un posible conjunto de valores para los parámetros. Cada uno de estos escenarios pueden tener una probabilidad asociada aunque no necesariamente (Dembo, 1991).

Existen diferentes modos de formular mediante un problema de Programación Lineal un Problema Estocástico aunque básicamente consiste en obtener una decisión para el instante actual teniendo en cuenta los escenarios futuros. De este modo la decisión a tomar no será óptima, en general, para ninguno de los escenarios aunque sí para el conjunto de ellos. Este modo de plantear y resolver el problema tiene un elevado coste computacional pero se puede abordado mediante descomposiciones y computación en paralelo con índice de paralelización elevado.

Otro modo de abordar la estocasticidad en los parámetros es obtener el óptimo para cada escenario y comparar el valor que esta decisión tendría para el resto de escenarios, eligiendo como decisión definitiva la más buena, o la menos mala o cualquier otro mecanismo que se considere oportuno.

#### *2.3.2.4 Programación No-Lineal*

Si a los modelos de Programación Lineal se les elimina el requerimiento de que la función objetivo o las restricciones sean lineales, se obtienen modelos de Programación No-Lineal. La eliminación del requerimiento de linealidad se suele fundamentar en la estructura no-lineal del objeto, o parte de él, a modelar. En realidad debiera ser planteado al revés, la realidad es generalmente no lineal y al linealizar estamos constriñendo el modelo.

Sin embargo muchas de las circunstancias aparentemente no-lineales pueden ser linealizadas sin pérdida de su significado. El motivo de la aparente obsesión por la linealidad se basa, fundamentalmente, en la falta de eficacia en la obtención de óptimos mediante el uso de los procedimientos actualmente existentes para la resolución de problemas no-lineales en general.

De hecho la no linealidad impide garantizar siempre la detección de un óptimo aún tras haberlo encontrado. El teorema de optimalidad **de Karush Kuhn y Tucker[4] (más conocidas por condiciones KKT)** que establecen las condiciones necesarias de optimalidad en problemas no lineales. Es de destacar que dichas condiciones no son suficientes sino necesarias.

Una variante de la programación No-Lineal la constituyen aquellos problemas sin restricciones, para los que el cálculo del óptimo tiene su origen en el desarrollo del cálculo matemático del siglo XVIII.

[4] Para los que tenemos una cierta edad, las condiciones son de Kuhn y Tucker. Kuhn y Tucker pretendían extender las bondades de la Programación Lineal a principios de los 50. Y publicaron sus condiciones. Según el propio Kuhn explica, cuando las probaron no sabían que Karush las había desarrollado como tesina fin de master en 1939 pero que nunca las había publicado, y que se dieron cuenta de ello 25 años después.

En general se puede admitir que la resolución de grandes problemas de optimización en programación No-Lineal aún hoy no es eficiente. Sin embargo el esfuerzo realizado no es infructuoso, por poner un ejemplo algunos de los algoritmos que actualmente permiten que la programación Lineal sea hoy en día tan rápida en su resolución fueron desarrollados para buscar soluciones en entornos No-Lineales. Posteriormente se han revelado eficientes en los problemas de Programación Lineal.

### 2.3.3 Los Componentes de un Modelo Matemático

Los modelos matemáticos tienen dos componentes básicos:

- Datos: Valores conocidos y constantes.
- Variables: Valores que se calculan.

Mediante la combinación lineal de los mismos se generan:

- Función Objetivo que debe minimizarse o maximizarse.
- Restricciones que establece límites al espacio de soluciones.

Tanto la función objetivo como las restricciones se expresan matemáticamente mediante el uso de variables o incógnitas. Se pretende definir unos valores a dichas variables de tal modo que se obtiene la mejor valoración de la función objetivo mientras se cumplen todas las restricciones.

En su formulación básica los modelos matemáticos tienen una función objetivo y una o más restricciones. Sin embargo existen excepciones como:

- Múltiples Objetivos
- Objetivos No existentes
- No existencia de restricciones

#### 2.3.3.1 Múltiples objetivos

Un modelo de Programación Matemática exige una única función objetivo que tiene que ser maximizada o minimizada. Esto sin embargo no implica que no se puedan abordar los problemas con múltiples funciones objetivo. De hecho, como se ha comentado, existen diferentes métodos de modelado y posterior resolución que se pueden aplicar en estos tipos de problemas.

Numerosos autores relacionan la Programación Multi-Objetivo con la Teoría de la Decisión que se aborda más adelante.

#### 2.3.3.2 Objetivos no existentes

En ocasiones al plantear el problema es difícil establecer un objetivo para el problema, más allá de encontrar una solución que satisfaga las restricciones. En ese caso es conveniente fijar un objetivo sencillo ligado a una única variable. Aunque la experiencia muestra una y otra vez, que una vez se

obtiene una solución factible el usuario acaba encontrando un modo de distinguir una solución de otra peor.

#### 2.3.3.3 Optimización sin restricciones

Los problemas de optimización sin restricciones pretenden minimizar (o maximizar) una función real $f(x)$ donde $x$ es un vector de $n$ variables reales. Es decir se buscan un $x^*$ tal que $f(x^*) \leq f(x)$ para todos los $x$ cercanos a $x^*$.

En el caso de un problema de optimización global, el $x^*$ buscado es el que minimiza $f$ para todo el espacio $x \in R^n$.

## *2.4 Modelos de Optimización Combinatoria*

La Optimización Combinatoria es una rama de la Investigación Operativa que consiste en encontrar la solución óptima a un problema en que cada solución está asociada a un determinado valor (el valor de la solución).

El término Combinatoria hace a la rama de la matemática que estudia colecciones finitas de objetos que satisfacen unos criterios especificados, y se ocupa, en particular, del "recuento" de los objetos de dichas colecciones (*combinatoria enumerativa*) y del problema de determinar si cierto objeto "óptimo" existe (*combinatoria extremal*).

El término Optimización hace referencia a este segundo aspecto de la búsqueda del mejor valor. En muchos de esos problemas la búsqueda exhaustiva no es factible y por la estructura de los problemas tanto no es posible.

La optimización combinatoria actúa en el campo de los problemas de optimización en los que el conjunto de soluciones factibles es discreto (o reducible a discreto). En algunos casos se tiende la tendencia a asumir que la OC es la programación lineal entera con variables binarias.

La búsqueda (o la definición de la existencia) de un óptimo para tipos específicos de problemas ha dado lugar a una cantidad considerable de algoritmos que son capaces de resolver casos específicos en tiempo polinomial.

Los problemas de optimización combinatoria tratan de encontrar la mejor solución de entre un conjunto de ítems discreto y por tanto finito. En principio cualquier algoritmo constructivo o de búsqueda podría encontrar la solución óptima, pero no necesariamente tiene porqué garantizarla.

En aquellos casos en que el problema parece resistirse a ser resuelto se suele abordar el problema de diferentes maneras:

- Algoritmos que funcionan bien generalmente para tamaños pequeños de problemas.
- En ocasiones hay problemas que su versión “aplicada” no presenta las características del *worst-case*.

- Algoritmos de aproximación que son capaces de dar soluciones muy cerca del óptimo.

## *2.5 Otros Modos de Modelar. Otros Modos de Pensar*

En el punto anterior se ha planteado una visión general de la Programación Matemática entendida en un sentido restrictivo. A continuación se revisan algunas técnicas diferentes en el fondo o en la forma. La lista, que no es exhaustiva ni las agrupaciones consideradas son necesariamente disjuntas, incluye las Teorías de Redes, de Colas y de Juegos, la Simulación, la Programación Dinámica. No se consideran, aunque son también importantes, modos de modelar como la Previsión (Companys, 1990), la Teoría de Decisión (White, 1972; Raiffa, 1970) o Teoría de Juegos (Binmore, 1994), o aplicaciones concretas como modelos de Inventario (Axsäter, 2000) o de Reemplazo (Figuera y Figuera, 1979).

### 2.5.1 Teoría de Grafos

Según Kauffman (Kauffman, 1972), la Teoría de Redes es una rama de la teoría de conjuntos basada en los trabajos de Köning. En aquel momento, era para el autor, la rama de la teoría de conjuntos con más futuro. De hecho aporta una ayuda eficaz para modelar y resolver determinados problemas de carácter combinatorio que aparecen en los más diversos dominios(Companys, 2003).

La teoría de redes, o de grafos, incluye un modo de representar y un soporte matemático para resolver. El modo de representar es intuitivo en su forma más simple, por su relación con la realidad física, de determinados tipos de problemas. El soporte matemático es especial para cada tipo de problema, que suelen ser complejos problemas de combinatoria, permite resolverlo de modo más simple que al utilizar la Programación Matemática convencional. Es relativamente sencillo traducir un modelo de red a un modelo de Programación Matemática, es un poco más complicado hacer la traducción a la inversa. La decisión sobre qué modo de modelar se debe utilizar, debe tomarla el modelador teniendo en cuenta la necesidad de transmitir el modelo (donde la teoría de grafos es claramente superior), la disponibilidad de herramientas y la realidad concreta a modelar.

En líneas generales se puede decir que los componentes básicos de la denominada Teoría de Grafos son los vértices (o nodos o puntos) y los arcos que los unen. A un conjunto determinado de vértices y arcos se le denomina “red”. A partir de estos conceptos se desarrollan otros como camino, corte, árbol, etc.

Algunos de los principales modelos que de este modo se plantean son: los problemas de árbol mínimo, de camino mínimo, de flujo máximo o de permutación óptima. El poder reducir un problema real a un problema clásico de grafos implica la posibilidad de conocer métodos eficaces de resolución, para muchos de ellos (siempre dependiendo del tamaño y la complejidad). Algunos de los problemas de Gestión Industrial que se pueden abordar con estos métodos son la programación de Proyectos, la Gestión de Inventarios, Diseño de Rutas, Secuenciación y Temporización, etc.

### 2.5.2 Programación Dinámica

Si antes se destacaba que el nombre de Programación Matemática no era muy representativo de la propia técnica, el de Programación Dinámica no lo mejora.

Cuando el nombre Programación Matemática había adquirido cierto auge, Bellman planteó en la década de los 50 (del siglo XX) la herramienta denominada Programación Dinámica, a través de su libro del mismo título para la resolución de problemas de carácter secuencial (inicialmente económicos pero también físicos y matemáticos).

El fundamento de este procedimiento se encuentra en el principio de optimalidad que Bellman enunció del siguiente modo:

> "Una política es óptima si en un periodo dado, cualesquiera que sean las decisiones precedentes, las decisiones que queden por tomar constituyen una política óptima teniendo en cuenta los resultados precedentes"

La Programación Dinámica es un método de optimización de los sistemas o de su representación matemática, sobre la que se opera por fases o secuencias (Kauffman, 1972).

El método de resolución denominado Programación Dinámica consiste en buscar las subpolíticas óptimas que comprendan cada vez más fases unitivas (Denardo, 1982), hasta encontrar la, o las, políticas óptimas. En ciertos problemas los cálculos se vuelven mucho más simples cuando se hace la optimización en un cierto sentido privilegiado o a partir de cierta fase (Companys, 2002).

Las variables utilizadas pueden ser discretas o continuas. Además los parámetros pueden ser deterministas o estocásticos. Cuando el futuro es aleatorio (parámetros estocásticos), la optimización de la esperanza matemática del valor total sólo puede llevarse a cabo en un sentido, remontándose desde el futuro hacia el pasado(Kauffman, 1972).

Si al concepto de Programación Dinámica se le une la consideración de los métodos de Ramificación y Corte, aparece el concepto de Programación Dinámica Acotada, por el cual se utilizan cotas en un esquema de Programación Dinámica, limitando el número de vértices que se pueden almacenar (Bautista, Companys, and Corominas, 1992)

Por las propias características de la aproximación a la resolución de problemas de Programación Dinámica (analizar el problema desde el final y retroceder por el camino hacia el principio) se puede compartir la afirmación de que la Programación Matemática, además de un modo de modelar es un "modo de vida".

### 2.5.3 Teoría de Colas

Se admite como inevitable la existencia de colas en los sistemas en que las entradas y/o el servicio se producen a intervalos irregulares. La teoría de colas es un método de modelado que describe el comportamiento de las mismas. La primera aplicación práctica de la que se tiene constancia, y con la que se inicia la investigación en este campo es el trabajo de Erlang a principios del siglo XX.

Uno de los resultados más conocidos de la teoría de colas es la denominada fórmula de Little que relaciona la longitud de la cola con el tiempo de espera y el ritmo de entrada al sistema.

Los resultados más habituales de la teoría de colas se refieren a sistemas de una etapa con entradas y salidas siguiendo distribuciones exponenciales. Sin embargo más útiles en múltiples ocasiones son las redes de colas con tiempos de los procesos no necesariamente exponenciales. Un excelente resumen de la situación actual de la teoría de colas se puede encontrar en (Gross, Shortle, Thomson, and Harris, 2008).

Los desarrollos en teoría de colas han ido extendiendo sus soluciones tanto para diferentes tipos de entradas como para redes de colas.

Es de destacar el especial interés que la teoría de colas tiene en el diseño de elementos estructurales de la denominada Nueva Economía (servidores web, procesadores compartidos...).

### 2.5.4 Dinámica de Sistemas

Se atribuye a Forrester el inicio del desarrollo de la denominada Dinámica de Sistemas. Esta tiene su relación directa con el Enfoque de Sistemas visto en el apartado dedicado a los Fundamentos Organizativos de la Organización de Empresas.

Forrester desarrolló un conjunto de herramientas y una aproximación a la simulación que se ha llegado a conocer como dinámica de sistemas, mediante la cual se puede llegar a comprender como la estructura de un sistema humano y sus políticas de control operan. Mostró también el valor que tienen los modelos explícitos que combinan procesos de negocio y estructura organizacional.

En (Pidd, 1996) se sugiere que el precursor de esta idea es Tustin que en 1953 publicó un trabajo titulado “El mecanismo de los sistemas económicos”.

Las herramientas de la Dinámica de Sistemas pueden utilizarse de diferentes maneras. La aproximación básica suministra una manera de observar sistemas humanos, haciendo especial hincapié en la importancia de algunos aspectos estructurales como el control por retroalimentación. La consideración de esta circunstancia aporta interesantes resultados incluso sin el uso de herramientas informáticas. Otro modo de utilizar la dinámica de sistemas es realizando simulaciones mediante ordenador, que permitan entender mejor el funcionamiento de otro sistema. Por último, la Dinámica de Sistemas se puede utilizar para mejorar el diseño de un sistema, evaluando mediante simulación su comportamiento.

### 2.5.5 Simulación

Asociada en ocasiones a la teoría de colas y heredera de la dinámica de sistemas se encuentra otra herramienta de los métodos cuantitativos como es la simulación. El incremento de la capacidad de cálculo de los ordenadores, así como sus crecientes capacidades gráficas hace que esta última esté experimentando una aplicación creciente en el modelado de flujos de materiales, e incluso de información.

Esta aplicación creciente ha supuesto, en algunos casos, el abandono de las herramientas analíticas, que requiere un esfuerzo conceptual que aparentemente la simulación no requiere. Hay que destacar, en contra de esta opinión que la simulación bien aplicada exige un importante esfuerzo para garantizar la validez de resultados. De hecho, dado que la simulación es comparable al análisis por experimentos, al hacer una simulación hay que hacer frente a los mismos problemas que hay que afrontar cuando se hace experimentación convencional (incluyendo diseño experimental y análisis estadístico). De este modo el uso de la simulación no reduce el esfuerzo a realizar, sino que resuelve problemas que la teoría de colas analítica no es actualmente capaz de abordar(Gross and Harris, 1998).

Pero no sólo la simulación de eventos discretos está disponible (aunque es con mucho la más utilizada en el terreno práctico) sino que la simulación basada en agentes y/o la simulación mediante Dinámica de Sistemas tiene su importante utilidad al modelar otros conceptos.

### 2.5.6 Teoría de Juegos

Algunos problemas de toma de decisión se plantean bajo la forma de un juego, donde se trata de tomar una o varias decisiones, frente a uno o varios decisores cuyas reacciones a las decisiones tomadas se conocen poco o nada. La teoría de juegos trata de establecer como debiera comportarse racionalmente un individuo ante la ignorancia del comportamiento del adversario cuando se conocen las reglas de la competencia aceptadas por los participantes (Kauffmann, 1972).

Para Kauffman (Kauffman, 1972), la teoría de juegos se desarrolla a partir de los trabajos de Borel (1921) aunque el crédito se suele asociar a Von Neumann (1924). Desde luego el concepto ya había preocupado en diferentes formas a Kepler, Huyghens, Pascal, Fermat y Bernouilli entre otros.

Von Neumann y Morguestern en su primera obra sobre Teoría de Juegos, investigaron dos planteamientos distintos: el estratégico o no-cooperativo y el cooperativo. La complejidad del primer problema planteado hizo que se limitaran a los juegos estrictamente competitivos o de suma nula, donde los dos jugadores (sólo dos) tienen intereses diametralmente opuestos. La segunda parte (el juego cooperativo) aún fue más complejo y por tanto se limitaron a clasificar los modelos de formación de coaliciones (Binmore, 1994).

Nash (año) afrontó dos de las barreras que Von Neumann y Morgenstern se habían autoimpuesto. Sobre los juegos no-cooperativos estableció la idea del equilibrio que aparece cuando la elección estratégica de cada jugador es la respuesta óptima a las elecciones estratégicas de los otros jugadores, con lo que no es necesario re*string*irse a los juegos de suma cero. Respecto a los problemas cooperativos, Nash rompió con la idea de que los problemas de negociación son indeterminados y ofreció argumentos para determinarlos (Binmore, 1994).

Actualmente la teoría de juegos, lleva aparejado un aparato matemático cada vez más complicado. En cualquier caso su modo de trabajo puede ser, y es, de gran utilidad en el análisis la toma de decisiones con la presencia de otros decisores.

# 3 PROGRAMACIÓN MATEMÁTICA

## 3.1 Introducción

Habría que poner un apartado de introducción en cada capítulo

## 3.2 La construcción de un Modelo de Programación Matemática

Es importante destacar que en la metodología que se explica a continuación el camino es de continuas idas y vueltas. Así lo normal es que una vez terminada una fase haya que volver a una etapa anterior y de este modo volver a comenzar.

Conjuntos de Datos, y por tanto de Índices. Conocer los tipos de datos de los que se dispone permite establecer, conjuntos y con ellos índices. Muchos de los que se definen en esta fase no son estrictamente necesarios, y otros se incorporarán en fases siguientes.

Parámetros. Representar los conjuntos de datos mediante Símbolos con subíndices, permitirá comenzar la conceptualización del problema. Generalmente en esta fase aparecen nuevos índices, o incluso se establecen parámetros que luego se comprobará que son variables.

Objetivo. Establecer la función objetivo en forma de lenguaje natural (Maximizar el beneficio esperado o minimizar el ratio de aprovechamiento) permite empezar a definir variables que se pueden denominar de control. No es importante en una primera fase establecer el objetivo de modo lineal, simplemente con representarlo ya es suficiente.

Variables de Control. De la etapa anterior se han definido las variables que configurarán la función objetivo. Dichas variables deben ser explícitamente representadas. En general, aunque algunos autores opinan lo contrario, se puede decir que las variables en la función objetivo no son las decisiones que se toman, sino los efectos de dichas decisiones, es por ello que se ha preferido denominarlas de control.

Restricciones. El modo más habitual de generar restricciones es expresarlas verbalmente y cuantificarlas posteriormente. Lo habitual es que surjan nuevos datos, parámetros índices y variables. En principio tras fijar las restricciones evidentes se observará que las variables de decisión y las de control no están conectadas. Las conexiones darán lugar a nuevas restricciones.

Variables de Decisión. En general al plantear la función objetivo no se está plasmando las decisiones que en realidad se quieren tomar. Dichas variables deben ser también reflejadas.

Modelo Completo. La construcción del modelo completo (funciones objetivo más restricciones) dará generalmente nuevos índices, parámetros, variables y restricciones.

Validación. La validación del modelo exige su formulación y su resolución. La importancia de esta etapa exige un apartado entero (el siguiente). La validación suele exigir recomenzar el proceso de validación desde el principio.

Por último convendría recordar que el buen modelo, su resolución y el análisis de los resultados, generalmente dará una mejor comprensión del problema, cerrando de este modo el círculo, pues generalmente una mejor compresión del problema, será *de facto* un nuevo problema. .

## *3.3 Implementación de un Modelo de Programación Matemática (Validación)*

Un modelo matemático sobre el papel es o debiera ser consistente. Ahora bien su introducción en un formato que permita resolverlo adecuadamente nos informará de nuevos errores que lleva.

Antes de prestar demasiada atención a la solución obtenida tras resolver un modelo que se haya construido, se debe comprobar que se ha modelado lo que se pretendía. Asumiendo que no hay errores de tipo sintáctico (generalmente los paquetes informáticos los detectan) existen tres salidas que en general se recorrerán sucesivamente: que el modelo sea incompatible, que el modelo no esté acotado y que el modelo sea resoluble. Aún llegado este punto caben dos situaciones, que el modelo sea resoluble pero sus soluciones no sean coherentes con el problema, y por último que el modelo sea resoluble y sus soluciones sean coherentes. Si se alcanza este punto, el modelo está listo para ser utilizado.

### 3.3.1 Modelo de sintaxis errónea

El error más típico es que la función objetivo no sea una fórmula sino un conjunto de fórmulas. Una manera de detectarlo es comprobar que ningún índice en las variables de la función objetivo se queda sin su sumatorio correspondiente.

Otro error típico se da en las restricciones. Todos los índices de todas las variables utilizadas deben estar en la expresión de la restricción. Ya sea en forma de sumatorios, ya sea en forma del rango en el que aplica la restricción (el para todo $\forall$ ).

Utilizar los paquetes de resolución para detectar problemas es lo más razonable en tiempo. Un modo de utilizarlos eficientemente es pedirles que escriban el modelo en formato .lp

```
38
39  minimize obj: sum{i in I, j in J} c[i,j] * x[i,j];
40  s.t. phi{i in I}: sum{j in J} x[i,j] <= 1;
41  s.t. psi{j in J}: sum{i in I} x[i,j] = 1;
42
```

```
Minimize
 obj: + 13 x(1,1) + 21 x(1,2) + 20 x(1,3) + 12 x(1,4) + 8 x(1,5)
 + 26 x(1,6) + 22 x(1,7) + 11 x(1,8) + 12 x(2,1) + 36 x(2,2) + 25 x(2,3)
 + 41 x(2,4) + 40 x(2,5) + 11 x(2,6) + 4 x(2,7) + 8 x(2,8) + 35 x(3,1)
 + 32 x(3,2) + 13 x(3,3) + 36 x(3,4) + 26 x(3,5) + 21 x(3,6) + 13 x(3,7)
 + 37 x(3,8) + 34 x(4,1) + 54 x(4,2) + 7 x(4,3) + 8 x(4,4) + 12 x(4,5)
 + 22 x(4,6) + 11 x(4,7) + 40 x(4,8) + 21 x(5,1) + 6 x(5,2) + 45 x(5,3)
 + 18 x(5,4) + 24 x(5,5) + 34 x(5,6) + 12 x(5,7) + 48 x(5,8) + 42 x(6,1)
 + 19 x(6,2) + 39 x(6,3) + 15 x(6,4) + 14 x(6,5) + 16 x(6,6) + 28 x(6,7)
 + 46 x(6,8) + 16 x(7,1) + 34 x(7,2) + 38 x(7,3) + 3 x(7,4) + 34 x(7,5)
 + 40 x(7,6) + 22 x(7,7) + 24 x(7,8) + 26 x(8,1) + 20 x(8,2) + 5 x(8,3)
 + 17 x(8,4) + 45 x(8,5) + 31 x(8,6) + 37 x(8,7) + 43 x(8,8)

Subject To
 phi(1): + x(1,1) + x(1,2) + x(1,3) + x(1,4) + x(1,5) + x(1,6) + x(1,7)
 + x(1,8) <= 1
 phi(2): + x(2,1) + x(2,2) + x(2,3) + x(2,4) + x(2,5) + x(2,6) + x(2,7)
 + x(2,8) <= 1
 phi(3): + x(3,1) + x(3,2) + x(3,3) + x(3,4) + x(3,5) + x(3,6) + x(3,7)
 + x(3,8) <= 1
 phi(4): + x(4,1) + x(4,2) + x(4,3) + x(4,4) + x(4,5) + x(4,6) + x(4,7)
 + x(4,8) <= 1
 phi(5): + x(5,1) + x(5,2) + x(5,3) + x(5,4) + x(5,5) + x(5,6) + x(5,7)
 + x(5,8) <= 1
 phi(6): + x(6,1) + x(6,2) + x(6,3) + x(6,4) + x(6,5) + x(6,6) + x(6,7)
 + x(6,8) <= 1
```

De este modo es posible visualizar si lo que hemos escrito en formato compacto se parece a lo que pretendíamos obtener en modo desarrollado.

### 3.3.2 Modelo incompatible

Un modelo no tiene solución si existen restricciones contradictorias. Por ejemplo, las siguientes restricciones harían el modelo irresoluble:

$$x_1 + x_2 \leq 1$$
$$x_1 + x_2 \geq 2$$

En la práctica esta situación no será tan evidente. El programa intentará resolver hasta que detecte la incompatibilidad. En algunos casos la indicará, aunque no es lo habitual.

En pocas ocasiones la incompatibilidad será debida a que el problema real no tiene solución, aunque en la mayor parte de casos es debido a que se ha formulado mal el problema.

Un modo lógico de buscar las restricciones que generan la incompatibilidad es ir anulándolas (anularlas aquí consiste en quitarlas del modelo para que no se tengan en cuenta o simplemente ponerlas como “comentario”). Hasta encontrar aquella (o aquellas) que al anularlas permiten que la solución exista. Hay que tener en cuenta que en general las restricciones son incompatibles a pares, por lo tanto no necesariamente la restricción anulada es la que está mal. Podría ser otra restricción que use las mismas variables. En pocas ocasiones, las incompatibilidades se producen por triangulación de restricciones (en pocas ocasiones pero ocurre).

Otro mecanismo para encontrar las incompatibilidades es generar una solución obvia (o real). Al obligarla, introduciéndola como restricciones de igualdad, el modelo debiera soportarla. Si no lo hicieran las restricciones violadas serán las incompatibles.

### 3.3.3 Modelos no acotados

Se dice que un problema no está acotado si su función objetivo se optimiza cuando alguna variable tiende a infinito. La solución obtenida en este caso, si puede ayudar a reconocer el error de formulación.

Mientras que en un modelo incompatible el problema es la existencia de demasiadas restricciones, en un problema no acotado faltan restricciones. Generalmente restricciones físicas obvias que han sido olvidadas o relaciones entre variables no consideradas. El ejemplo más típico es minimizar costes de inventarios en un problema de planificación de producción y olvidarse de imponer que la variable nivele de inventario es una variable positiva.

Los modelos no acotados tienen un “dual” incompatible, aunque la inversa no es siempre cierta. Por eso muchas herramientas resuelven inicialmente el dual.

### 3.3.4 Modelo Resoluble

Cuando un modelo no es incompatible y es acotado se conoce como un modelo resoluble. Cuando se obtenga la solución óptima de un modelo resoluble lo primero que hay que hacer es comprobar que ésta es lógica. Si no lo fuera el modelo sería erróneo. El sentido común es el mejor aliado en este caso.

Para comprobar que los resultados son lógicos existen múltiples maneras, y cada modelador tiene su metodología para el análisis de los resultados. Una que da buenos resultados es modificar de manera sistemática los costes en la función objetivo (o las limitaciones en las restricciones) para comprobar que se puede modificar la solución modificando los coeficientes.

Hay que destacar el valor de la optimización en la validación del modelo. Cuando se trata de maximizar (o minimizar) alguna cantidad, alguna restricción debe ser utilizada completamente. Es interesante, cuando se trata de un modelo nuevo, probar con diferentes objetivos, porque dan relevancia a diferentes restricciones y, por tanto, se pueden ir comprobando todas.

Si el resultado es lógico deberíamos compararlo con un resultado real, podría ser que el modelo fuera demasiado restrictivo (aunque no llegue a ser incompatible) o demasiado laxo (aunque no llegue a ser no acotado).

En muchos casos, este análisis ya está conduciendo hacia un mejor entendimiento del problema, y por tanto, el esfuerzo (aunque sin solución) ya está siendo eficaz.

De la exposición anterior se desprende de modo natural que el proceso modelado-validación debe ir repitiéndose hasta converger en un modelo que represente de modo suficientemente acertado la realidad. Y este proceso es muy útil para entender la propia realidad modelada.

## *3.4 Características de un buen modelo de Programación Matemática*

Tres son los objetivos básicos que se deben pretender al construir modelos de programación matemática.

- Que sea un modelo fácil de entender
- Que sea un modelo cuyos errores sean fáciles de detectar
- Que sea un modelo de fácil resolución

### 3.4.1 Facilidad para entender el modelo

A menudo es posible construir un modelo compacto y realista donde diferentes variables aparezcan de modo implícito y no de modo explícito. Sin embargo construir este tipo de modelos conduce habitualmente a soluciones difíciles de evaluar y más aún de interpretar. Incluso, aunque los modelos menos compactos requieren un tiempo más largo de resolución, este tiempo compensa el que posteriormente habría que invertir en la interpretación de la solución.

Es útil también utilizar acrónimos para designar las variables y las restricciones, de tal modo que los resultados se puedan, posteriormente, interpretar más fácilmente.

Existe también unas normas de estilo que, no siendo obligatorias permiten entender mejor el modelo, si todo el mundo las utiliza igual. En general se admite que las letras de la h a la l (minúscula) se refieren a índices, que las letras *x,y,z,v* son variables mientras que *a,b,c,d,m,n,p,q* suelen utilizarse como parámetros, o las letras griegas $\alpha,\beta,\mu,\pi,\delta,\lambda$ suelen hacer referencia a variables binarias. Aunque no existe ninguna norma escrita, el uso de este estilo simplifica el trabajo que supone entender un modelo. Otra forma de no confundirse es usar letras mayúsculas para los datos y letras en minúscula para las variables (aunque a veces, se hace al revés).

En cualquier caso es imprescindible que no existan homónimos ni sinónimos, esto es, elementos con el mismo nombre y significado distinto, o elementos con distinto nombre y el mismo significado. Y por otra parte, es muy importante usar una escritura estructurada simple para facilitar el entendimiento del modelo.

### 3.4.2 Facilidad para detectar errores en el modelo

Este objetivo está claramente unido al anterior. Los errores podrían ser de dos tipos: de tecleo (números erróneos o defectos ortográficos) y de formulación.

Para evitar los problemas de tecleo es útil utilizar programas de generación de matrices o lenguajes de modelado. Muchos de estos programas tienen procedimientos que detectan errores además de otras funciones relacionadas.

### 3.4.3 Facilidad para encontrar la solución

Los modelos de Programación Lineal utilizan una gran cantidad de tiempo de computación, y por tanto sería interesante construir modelos que sean resueltos de modo rápido. Este objetivo se contrapone en la práctica al primero de los enunciados. En algunos casos es posible reducir el tamaño del modelo mediante procedimientos instalados en el propio editor de modelos. En otros es un ejercicio que debe realizar el modelador.

A continuación se presentan tres métodos que permiten reducir el coste de computación: Uso de Cotas, Unidades de Medida apropiadas y uso de formulación modal.

#### 3.4.3.1 Uso de Cotas

Es posible alcanzar reducciones sustanciales en el tiempo de computación teniendo en cuenta las cotas globales (GUB) citadas en el capítulo siguiente. Evidentemente esta reducción será efectiva dependiendo de si el paquete lleva o no incorporadas rutinas que traten estas cotas.

#### 3.4.3.2 Unidades de medida

Cuando se modela una situación práctica es importante prestar atención a las unidades en las cuales se miden las cantidades. Si existe una gran disparidad en los coeficientes de un modelo de Programación Lineal se incrementa sustancialmente el tiempo de computación.

Es decir, si las restricciones de capacidad se miden en toneladas, los beneficios no se deberían dar en pesetas por kilo. Idealmente las unidades se deberían plantear de tal manera que los coeficientes no nulos estén entre 0.1 y 10. Algunos paquetes comerciales tienen procedimientos para “escalar” automáticamente los coeficientes antes de resolver y después “desescalar” también de modo automático.

#### 3.4.3.3 Formulación Modal

En problemas de Programación Lineal grandes, se puede reducir sustancialmente el número de variables utilizando la denominada formulación modal. Si un conjunto de restricciones se ve afectada por un reducido número de variables, se puede considerar como espacio de soluciones el espacio que marcan estas variables y sus restricciones. Y, de este espacio, sólo serían de interés algunos puntos (los vértices).

# 4 MODELOS DE PROGRAMACIÓN LINEAL

## 4.1 ¿Qué es la Programación Lineal?

El problema básico de Programación Lineal consiste en minimizar una función objetivo lineal de variables lineales continuas, sujetas a (s.a.) restricciones lineales. Se considera que una función es lineal si es una combinación lineal de las variables consideradas.

Una expresión estándar de un problema de Programación Lineal es:

$$\begin{aligned} &[Minimize] \quad \sum_i c_i \cdot x_i \\ &Sujeto\,a: \\ &\qquad \sum_i a_{i,j} \cdot x_i \geq b_j \quad \forall j \\ &\qquad x_i \geq 0 \quad \forall i \end{aligned}$$

La región de factibilidad es un poliedro y al menos una solución óptima es un vértice del citado poliedro.

Aunque la versión estándar del problema de Programación lineal es de minimización , los problemas pueden ser también de maximización. Que las variables tengan un solo subíndice es porque finalmente todo se puede reducir al uso de un único subíndice, pero la mayor parte de los problemas utilizan variables con más de un índice.

*Una empresa fabrica únicamente dos productos: P y Q.P se vende a 45 € y Q a 50 €. La demanda semanal de cada producto es de 200 unidades de P y Q=100 unidades de Q*

*Los dos productos requieren de una misma pieza central, la materia prima de la cual vale 20€ por unidad. Para fabricar la pieza central hacen falta 15 minutos del recurso B y 5 minutos del recurso C. Para fabricar el componente 1 del producto P hace falta materia prima por valor de 20€/unidad, 15 minutos del recurso A y 10 minutos del recurso C. Al ensamblar la pieza central con el componente 1 utilizamos otro componente 3 que se compra al precio de 5€/unidad, lo ensambla el recurso D en 15 minutos cada unidad.*

*El producto Q sigue un procedimiento similar. El componente 2 utiliza Materia Prima por valor de 20€/unidad, pasa por el recurso A donde está 10 minutos y luego por el proceso B donde está 15 minutos. Finalmente es ensamblado por el recurso D en 5 minutos. El mes tiene 20 días de 8 horas. Los gastos totales son 3600 €/semana.*

$$[Maximize] \quad \sum_i PV_i x_i - CC_j y_j$$

$$Sujeto\, a:$$

$$\sum_i R_{i,k} \cdot x_i \le KAP_k \quad \forall k$$

$$\sum_i x_i \le D_i \quad \forall k$$

$$y_j = \sum_i N_{i,j} x_i \quad \forall i$$

$$x_i \ge 0 \quad \forall i$$

## 4.2 La importancia de la Linealidad

Los modelos de Programación Lineal exigen que tanto las expresiones de las restricciones como de los objetivos se hagan de modo lineal. Esto quiere decir que no pueden aparecer expresiones del tipo $x_1 \cdot x_2$, $e^{x_1}$ ó $x_1^3$. Para muchos problemas prácticos esta es una consideración que impide el uso de la Programación Lineal (aunque existen modos de reconvertir modelos de Programación no-Lineal en modelos de Programación Lineal).

La principal razón por la que los modelos de Programación Lineal son preferibles es porque son mucho más fáciles de resolver. Los modelos de Programación Lineal son más fáciles de resolver puesto que en el espacio de soluciones factibles siempre será posible encontrar una solución en un vértice, reduciendo de este modo el espacio de posibles soluciones óptimas a un número finito de las mismas.

Desde luego en cada ocasión hay que comprobar que es posible resolver mediante Programación Lineal, es decir, comprobar que el modelo se ajusta suficientemente a la realidad.

## 4.3 Situaciones que pueden modelarse mediante Programación Lineal

La programación lineal, pese a sus restricciones evidentes por las características de las variables y las funciones, tiene una amplia variedad de aplicaciones.

Algunos de los problemas clásicos de Programación Lineal son:

- Blending (Mezcla). Este tipo de problemas considera las decisiones relativas a la mejor generación de uno o varios productos resultado de una mezcla. Quizá el problema más antiguo se refiere a la generación de dietas, pero también en la industria de elaborados alimenticios o en la industria petrolera tiene aplicaciones.
- Product Mix (Catálogo de Productos). En su versión más básica pretende establecer la proporción de productos a fabricar dados unos recursos limitados y unos beneficios esperados.

- Decisión de Inversiones. Otro de los problemas clásicos consiste en seleccionar la mejor cartera de inversiones minimizando riesgos o maximizando beneficios dado un conjunto de limitaciones en los recursos.
- Problema del Transporte. Consiste en definir, en una red de suministro, los centros que deben producir, la cantidad a producir y el destino de esta producción, teniendo en cuenta costes de transporte y/o de almacén, además de beneficios esperados y limitaciones en los recursos disponibles.

## *4.4 Los parámetros*

Quizá uno de los aspectos más curiosos para el modelador novel son los parámetros de los modelos. No se sabe exactamente se suele asumir que los parámetros del modelo existen, son ciertos y son conocidos.

El modelador inexperto suele tener la conciencia de que los datos están dados. El modelador quiere creer que valores como el coste o la duración de un cambio de partida, el ritmo de producción o su coste asociado, la capacidad de la máquina (¿medida en horas, en unidades?), el precio de los productos, o el nivel de stock en un momento dado para un determinado producto, es conocido.

Esto no es así en prácticamente ninguna empresa para casi ningún datos. Comenzando por la demanda que siempre es incierta por sus propias características, siguiendo por los costes que dependen del modo de cálculo y que en nuestro caso son siempre de oportunidad y por tanto inaprensibles antes de tener la solución al problema, pero también para parámetros que podrían ser más objetivos como los niveles de stock o los ritmos de producción en general ni están ni se les espera en ningún sistema de información que se precie.

Haría bien cualquier modelador en dudar, hasta el último momento de la calidad de los datos de partida. Lo que no haría bien es en dudar de la bondad de estimarlos y trabajar como si existieran. En algunos casos la mera existencia de un sistema que los utilice permitirá reducir la incertidumbre de los mismos. En otros casos el sistema es poco sensible al dato concreto, y un ajuste (*tuning)* de la primera aproximación permitirá obtener resultados más que suficientes.

Por último, si sigue persistiendo la duda respecto a la validez de los datos… es en ese momento cuando el modelado matemático más sentido adquiere. Pues permite, mediante la resolución iterativa de problemas, hacer un análisis de sensibilidad que mejore (y mejora) la percepción que se tiene del problema.

Pero también hay métodos de considerar explícitamente la incertidumbre en los parámetros como la programación estocástica o el uso de Programación Fuzzy.

## *4.5 Definición de Objetivos (por acabar)*

Dado un conjunto de restricciones, diferentes funciones objetivo conducirán a diferentes soluciones. Si diferentes funciones objetivo dan lugar a la misma solución es conveniente analizar la estructura del modelo puesto que puede aportar un mejor entendimiento del problema.

Algunos de los objetivos que se pueden definir son: Maximizar beneficio, Minimizar costes, Maximizar rentabilidad, Maximizar el número de empleados, Minimizar el número de empleados, Maximizar la satisfacción del cliente y Maximizar la probabilidad de sobrevivir.

Podrían definirse muchos otros objetivos. Pero también podría ocurrir que no se quisiera optimizar nada o que se quieran "optimizar" diferentes objetivos de modo simultáneo, teniendo estos objetivos direcciones conflictivas. En cualquiera de estos casos la Programación Matemática es interesante y debe ser utilizada.

### 4.5.1 Problemas Mono Objetivo

El caso de los Objetivos Simples es el caso más sencillo, generalmente se pretenden maximizar beneficios o minimizar costes. La principal ventaja de estos objetivos es que los diferentes factores se ven reducidos a una unidad común: la monetaria.

Si se pretende minimizar costes es importante fijar restricciones adecuadas, puesto que en ocasiones minimizar costes supone no hacer nada. También hay que distinguir de manera adecuada los costes fijos y los costes variables.

Si se pretende maximizar beneficios hay que tener en cuenta que estos se pueden obtener a lo largo del tiempo. Incorporar el concepto de tiempo en la evaluación del beneficio se puede hacer de múltiples maneras entre las que destaca el Valor Actual Neto.

### 4.5.2 Programación Multi-Objetivo

La Programación Multi-Objetivo es Programación Matemática, puesto que conceptualmente pertenece a ella con la característica de la existencia de más de un objetivo (lo que por otro lado es más habitual en la realidad que el caso de un solo objetivo).

La optimización Multi-Objetivo tiene sus raíces en los trabajos sobre economía de Edgeworth y Pareto realizados a finales del siglo XIX.

El concepto escalar de optimalidad no es aplicable en la Programación Multi-Objetivo, dando lugar al concepto de solución no dominada u óptimo de Pareto (recibe también otros nombres). Este concepto permite la existencia de múltiples soluciones óptimas. Existen algunos procedimientos eficaces como el simplex multiobjetivo, pero sólo son útiles para problemas de reducidas dimensiones.

**El método de las ponderaciones.**

Un modo de abordar este tipo de problemas es optimizar la suma ponderada de los valores de cada objetivo. Para algunos autores esto no es posible salvo que se encuentre un factor común como podría

ser el coste monetario, aunque esto no es siempre posible. Los pesos deben ser relativos (o valores monetarios o similares) y deben tener en cuenta el rango de acción de cada variable.

Al sumar diferentes objetivos, aunque sean ponderados, se está introduciendo un cierto grado de arbitrariedad por lo que será necesario comprobar que el resultado se ajusta a nuestros requerimientos.

Uno de los pesos que se pueden utilizar inicialmente son los precios-sombra[5] que en cada resolución dan las restricciones, para un modelo en que todos los objetivos menos uno se han convertido en restricciones.

Para que la resolución sea eficiente las ponderaciones deben ser todas positivas o todas negativas. El modo de ajustar los pesos (*parameter tuning*) puede alcanzar niveles de sofisticación muy elevados, aunque también se pueden ajustar *manualmente.*

*Método de las restricciones*

Otro modo interesante de abordar estos problemas es el denominado método de restricciones. En ella los objetivos se ordenan por importancia y se van tratando de optimizar uno detrás de otro, fijando en cada caso como restricción que el valor de las funciones objetivos "más" importantes ha de ser igual al óptimo encontrado. Una variante relajada de la anterior es la técnica denominada "*Goal Programming"* donde para cada objetivo se establece un valor suficiente (o valor meta), se fija también una penalización para el caso de que no se alcance dicho valor meta, y cada uno de los objetivos pasa al conjunto de restricciones pasando a formar parte de las restricciones. De este modo, al igual que en los casos anteriores el problema pasa a tener un solo objetivo, que es minimizar las penalizaciones ligadas a no alcanzar las metas fijadas.

Por último es posible abordar el problema Multi-Objetivo encontrando las superficies que forman los puntos no dominados, siendo el decisor el que selecciona su opción. Una solución no-dominada es aquella que al compararse con cualquier otra solución factible tiene al menos uno de los objetivos mejor.

Una primera aproximación consistirá en resolver el problema tantas veces como objetivos tenga el resultado, utilizando en cada ocasión un objetivo distinto. La comparación de los diferentes resultados puede dar una idea de qué solución es mejor. En este caso se puede utilizar la solución para un objetivo como la solución inicial.

Por último en ocasiones no existe ningún objetivo real, lo único que se pretende es encontrar una solución válida. En otras ocasiones el objetivo a perseguir no es optimizable (por ejemplo el objetivo "sobrevivir"). El uso de los modelos matemáticos nos puede permitir encontrar soluciones factibles (que cumplen las restricciones si éstas existen).

[5] El concepto de precio-sombra aparece al realizar la denominada interpretación económica de las soluciones y se presenta más adelante.

### 4.5.3 Programación Fuzzy Multiobjetivo

### 4.5.4 Programación Multinivel

## *4.6 Las restricciones*

### 4.6.1 Tipos básicos de Restricciones en Dirección de Operaciones

Las restricciones son expresiones de relaciones entre variables. Dichas relaciones se representan mediante restricciones en la programación matemática, y tienen la formulación de una combinación lineal de variables limitada por un determinado valor.

Las restricciones se pueden clasificar en función de la realidad que pretenden representar, o en función de su relación con el resto del modelo matemático.

Según su relación con la realidad que pretenden representar se pueden encontrar las siguientes:

- Restricciones de capacidad. Se limita el consumo de capacidad productiva de un conjunto de recursos/productos/operaciones.
- Disponibilidad de Materia Prima. Se limita el consumo de un determinado conjunto de productos (y en consecuencia la producción de un conjunto de productos) según la cantidad de materia prima disponible en cada momento.
- Limitaciones en la demanda del mercado. Se limita la producción de un producto en función de la venta estimada de éste.
- Restricciones de Continuidad o Balance de Materiales o Energía. En programación multiperiodo, los productos que quedan al final de un periodo son las que hay al principio del siguiente. También si un producto se descompone en otras unidades la suma de las cantidades descompuestas es igual a la cantidad original (o con un factor de rendimiento). Son también restricciones de continuidad las que conectan los diferentes arcos que entran o salen de un nodo (por ejemplo en problemas de distribución de energía).
- Estipulaciones de Calidad. Al planificar la producción de productos, se pueden establecer restricciones en función de las características de calidad de la mezcla y de las materias primas.
- Relaciones de tipo lógico. En ocasiones las restricciones tienen forma de expresión lógica, “si consumes más de 40kw, hay que activar un segundo generador”.

### 4.6.2 La relación de las restricciones con la realidad, con las otras restricciones y con el propio modo de resolver

Las anteriores restricciones forman parte de las relaciones concretas entre las variables de los problemas. También se podrían clasificar las restricciones en función de su comportamiento en el resto del modelo.

- Restricciones Duras y Blandas
- Restricciones Conflictivas
- Restricciones Redundantes
- Cotas Simples y Generalizadas
- Restricciones de Rango

**Restricciones Duras y Blandas**

Una restricción del tipo:

$$\sum_j a_j \cdot x_j \leq b$$

elimina cualquier solución para la que la suma sobre *j* de la variable *x* exceda el valor de *b*.

Esto puede ser considerado como poco realista en algunas ocasiones. Por ejemplo, si *b* son las horas disponibles quizá, si interesara, habría que contratar algunas horas extras. En este caso la restricción se denomina blanda. Son duras aquellas restricciones que no se pueden violar de algún modo.

Un mecanismo para modelar las restricciones blandas podría ser:

$$\sum_j a_j x_j - u \leq b$$

donde *u* es una variable que aparecería en la función objetivo con un coste *c·u*. De este modo en caso de ser necesario la restricción sería violada aunque penalizando la función objetivo. A la variable *u* se le puede incorporar un límite de tal manera que no exceda un cierto valor.

Si la restricción blanda es una desigualdad del tipo mayor o igual se puede aplicar el mismo esquema:

$$\sum_j a_j x_j \geq b$$

Si la restricción es una igualdad la restricción debe sustituirse por:

$$\sum_j a_j x_j + u - v = b$$

tanto *u* como *v* se verán penalizados en la función objetivo y en la solución uno de los dos valores debe ser cero.

**Restricciones conflictivas**

En ocasiones ocurre que un problema incorpora un conjunto de restricciones que no siempre pueden satisfacer al máximo cada una de las restricciones. En este caso los objetivos serían también conflictivos.

El tipo de modelo a que da lugar esta situación es de los que se llaman modelos de "coordinación de objetivos". Cada restricción es, en este caso un objetivo, que debe cumplirse tanto como sea posible.

Por ejemplo, si se pretendiera imponer el siguiente conjunto de restricciones:

$$\sum_j a_{ij} \cdot x_j = b_i \qquad \forall i$$

Y se deseara admitir que no todas se cumplen, se podría modelar del siguiente modo:

$$\sum_j a_{ij} \cdot x_j + u_i - v_i = b_i \qquad \forall i$$

El objetivo sería asegurar que cada restricción se cumple lo máximo posible. Este objetivo se puede modelar, entre otros modos de estas dos maneras

a) Minimizar la suma de todas las desviaciones

$$[Minimize] \ \sum_i (u_i + v_i)$$

b) Minimizar la máxima desviación

$$\begin{aligned} &[Minimize] \quad z \\ &Sujeto\,a: \\ &\qquad z - u_i \geq 0 \qquad \forall i \\ &\qquad z - v_i \geq 0 \qquad \forall i \end{aligned}$$

Sea cual sea el método elegido podría a su vez ser ponderado si alguna de las restricciones parece más importante.

**Restricciones redundantes**

En el caso en que se tenga una restricción del tipo:

$$\sum_j a_j \cdot x_j \leq b$$

y la evaluación $\sum_j a_j \cdot x_j$ sea en cualquier caso inferior a *b*, se puede decir que la restricción es redundante, o que su precio sombra es nulo, es decir no tiene influencia sobre la función objetivo.

Una restricción redundante puede ser eliminada sin afectar al óptimo. Generalmente no es posible eliminar las restricciones redundantes a priori. Además, si el modelo se va a usar de modo continuado, la restricción no se puede eliminar por si nuevos valores de los coeficientes convierten a la restricción en relevante.

Hay que tener en cuenta que en la Programación Entera la redundancia no es tan fácil de considerar como en la Programación Lineal, más aún en ocasiones las restricciones redundantes facilitan la búsqueda del óptimo.

**Cotas Simples y Generalizadas**

Las cotas simples se pueden manejar de manera mucho más eficiente definiéndolas como tales. El motivo, que ya se ha indicado anteriormente es la existencia de una versión revisada del Simplex que maneja de manera especial estas cotas y reduciendo el tiempo total de computación.

Las cotas simples tienen las siguientes formulaciones:

$$x \le U \quad \text{o} \quad x \ge L$$

Las otras cotas que algunos paquetes de resolución consideran son las denominadas Cotas Generalizadas (GUB):

$$\sum_i x_i \le U$$

**Restricciones de Rango**

En ocasiones las restricciones adquieren este formato:

$$b_2 \le \sum_j \left(a_j \cdot x_j\right) \le b_1$$

Estas restricciones (en realidad son dos) pueden expresarse como tales, pero existe un modo más compacto y que está disponible en muchos paquetes comerciales. Consiste en definir únicamente la restricción:

$$\sum_j a_j \cdot x_j \le b_1 \qquad \text{con un rango de } (b_1\text{-}b_2)$$

Otro modo de expresarlas consiste en transformar las restricciones en otras del siguiente tipo:

$$\sum_j \left(a_j \cdot x_j\right) + u = b_1$$
$$0 \le u \le b_1 - b_2$$

siendo estas restricciones más fáciles de manejar por la mayor parte de paquetes y, por tanto, el tiempo de computación para obtener la solución se verá reducido.

## 4.7 Análisis de resultados

### 4.7.1 Interpretaciones económicas

Para avanzar en esta sección es conveniente plantear un problema de mezcla de productos que sirva como ejemplo.

*Una empresa fabrica 5 tipos de productos utilizando fundamentalmente 2 tipos de recursos (3 tornos y 2 fresas). Tras deducir el coste de la materia prima cada producto aporta el beneficio expresado en la tabla 2. En la misma tabla se representa los tiempos de operación necesarios para hacer cada producto en cada máquina.*

| | Pr 1 | Pr 2 | Pr 3 | Pr 4 | Pr 5 |
|---|---|---|---|---|---|
| Torno | 12 | 20 | - | 25 | 15 |
| Fresa | 10 | 8 | 16 | - | - |
| Beneficio unitario | 550 | 600 | 350 | 400 | 200 |

*La semana tiene 6 días de 16 horas.*

*Al acabar el proceso de fabricación es necesario proceder al montaje final de cada producto que exige 20 horas de operario. La empresa dispone de 8 operarios trabajando en cada uno de los dos turnos.*

*El modelo del programa es:*

*[Maximize] 550 x1 +600 x2 + 350x3 + 400x4 + 200x5*

*Sujeto a:*

| | | | | | | |
|---|---|---|---|---|---|---|
| Tornos | $12x_1$ | $+20x_2$ | | $+25x_4$ | $+15x_5$ | $\leq 288$ |
| Fresas | $10x_1$ | $+8x_2$ | $+16x_3$ | | | $\leq 192$ |
| Montaje | $20x_1$ | $+20x_2$ | $+20x_3$ | $+20x_4$ | $+20x_5$ | $\leq 384$ |

*La solución del problema es*

$$x1=12;x2=7,2;x3=x4=x5=0$$

*Con valor de la función objetivo de 10920 euros*

***Ejemplo 1***

De la observación de los resultados se conoce que tanto la restricción de tornos como la de Montaje están al límite, mientras que fresas tiene aún capacidad.

Pero además existe información de interés que puede salir del modelo.

a) ¿Cuánto debieran incrementar su precio los productos 3,4,5 para que fueran rentable fabricarlos (coste reducido)?

b) ¿Cuánto vale una hora extra de cada recurso (precio-sombra)?

Estos resultados pueden obtenerse del análisis de los resultados que ofrece el simplex si este método ha sido utilizado como el de resolución. También pueden obtenerse de la resolución del dual, y en cualquier caso cualquier herramienta de resolución actual ofrece incorpora un análisis de sensibilidad más que suficiente para conocer tanto los costes reducidos y los precios sombra, como los rangos en los que se deben mover las variables para no cambiar de solución básica.

**Precios sombra**

El "precio sombra" es el precio máximo que se debería pagar por una unidad extra de un determinado recurso. Estas variaciones son siempre marginales y tienen un cierto límite (incorporar 10000 horas de montaje sería absurdo). Estos límites (rangos) se establecen en la siguiente sección.

Los "precios sombra" de varias restricciones no pueden utilizarse simultáneamente pues las variaciones en los términos independientes de varias restricciones simultáneamente, no suponen la suma de los precios sombra en la función objetivo. Los precios sombra son un claro ejemplo de costes de oportunidad, concretamente del coste de la oportunidad perdida.

**Costes reducidos**

Los "costes reducidos" asociado una variable son el valor que habría que subir el precio (o la contribución marginal) para cada valor de sus índices correspondientes para que la variable que lo representa no fuera nula. También este incremento en precio tiene un límite por encima del cual no se van a establecer más variaciones. Otra manera de observar los costes reducidos es como el dinero que se dejaría de ganar si se obliga al sistema a producir al menos un producto del tipo en cuestión.

### 4.7.2 Análisis de Sensibilidad

Como se ha comentado anteriormente los parámetros de los modelos no tienen porqué ser eactos y ciertos. Es por ello que al btener una solución interesa saber cómo se hubiera comportado la solución en caso de que los datos hubieran sido ligeramente diferentes. Ese análisis (fruto fundamental de cualquier resolución) se puede realizar resolviendo una y otra vez el mismo modelo, pero también se puede obtener analizando ligeramente algunos datos más que suelen obtenerse al resolver otros modelos

Al resolver un modelo lo importante generalmente no es conocer los valores solución de cada una de las variables, o el valor de la función objetivo sino la política a la que conduce el modelo. Así en el modelo de ejemplo (ejemplo 1) lo que importa es que se fabrican productos de tipo 1 y tipo 2 de tal manera que utilizan a plena capacidad los recursos torno y mano de obra. Si un modelo es estable ligeras variaciones de los coeficientes no variarán esta política, sino únicamente las cantidades concretas.

**Rangos en las restricciones**

En el apartado anterior se han descrito los “precios sombra” como útiles para predecir el efecto de pequeños cambios en el valor de la restricción. Se ha indicado también que estos cambios son así interpretables dentro de un determinado rango.

Conocer los rangos en los que se puede mover una restricción sin alterar gravemente la solución, sirve para realizar análisis de sensibilidad. En ocasiones los valores de las restricciones (y otros coeficientes) no son estrictamente conocidos, pero si la solución no variara (conceptualmente) con variaciones en los coeficientes dentro del rango, es útil conocer éste para saber si hay que invertir un mayor esfuerzo en conocer exactamente el dato.

Cuando los rangos son amplios se puede asumir que un modelo es estable.

**Rangos en el objetivo**

A menudo es útil conocer los efectos de cambios en los coeficientes de la función objetivo en la solución óptima (para ello se han definido los costes reducidos). Análogamente a las restricciones se pueden definir rangos de variación donde no existirán variaciones en la estructura de la solución.

La misma aplicación en análisis de sensibilidad que los rangos en las restricciones pueden tener los rangos en los objetivos. Además de que aquí no solo variará la “política” de la solución sino tampoco la cantidad exacta.

### 4.7.3 El Modelo Dual

El modelo dual se genera al convertir los objetivos en restricciones y las restricciones en objetivos. Básicamente se trata de crear tantas variables como restricciones se encuentran, y combinarlas linealmente con el término independiente de cada restricción para generar la función objetivo. Al mismo tiempo los coeficientes de la función objetivo pasan a ser los términos independientes de las restricciones, cuya matriz tecnológica es la transpuesta de la original.

De este modo el problema del ejemplo XXXX queda de la forma siguiente:

[Minimize] $288y_1+192y_2+384y_3$

Sujeto a:

| | | | |
|---|---|---|---|
| $12y_1$ | $+10y_2$ | $+20y_3$ | $\geq 550$ |
| $20y_1$ | $+8y_2$ | $+20y_3$ | $\geq 600$ |
| | $16y_2$ | $+20y_3$ | $\geq 350$ |
| $25y_1$ | | $+20y_3$ | $\geq 400$ |
| $15y_1$ | | $+1y_3$ | $\geq 200$ |

Al resolver el modelo dual se obtiene la siguiente solución para las variables:

$y_1=6,25 \quad y_2=0 \quad y_3=23,75$

***Ejemplo 2***

Esto significa que si aumentáramos en una unidad el número de horas disponibles de torno el incremento en el beneficio de la empresa sería de 6,25 €, mientras que sería nulo si aumentáramos las horas de fresa. Este concepto es el denominado Precio Sombra.

### 4.7.4 Modelos Estables

En muchas ocasiones es más interesante encontrar soluciones estables que soluciones óptimas. Las soluciones estables son aquellas en las que pequeñas variaciones de los coeficientes no alteran la solución final. Esto se da en una gran proporción de casos en los que no se puede cambiar la política de una empresa debido a pequeñas variaciones en los datos de entrada.

De algún modo este tipo de situación puede ser modelado “relajando” algunas restricciones, es decir, suponer que se va a poder violar la restricción:

$$\sum_j a_j \cdot x_j \leq b$$

Para ello resulta de interés que se pueda establecer como:

$$\sum_j a_j \cdot x_j - u = b$$

Y un sumando en la función objetivo que penalice el uso de *u*.

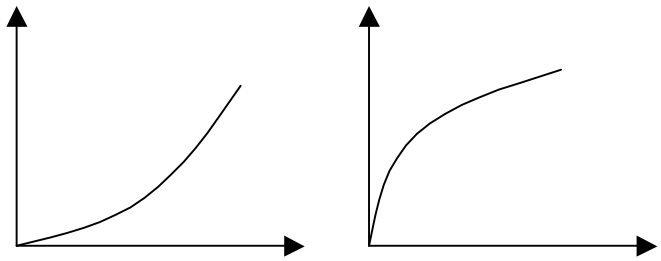
# 5 LINEALIZANDO LO NO-LINEAL

Cuando se trata de un modelo lineal los beneficios (o costes) son proporcionales a la cantidad vendida o fabricada. El uso de un recurso limitado depende linealmente de la cantidad fabricada y el uso total de un recurso. Este uso de un recurso es la suma de los consumos parciales del conjunto de actividades que se ejecutan en él.

Todas estas expresiones configuran un modelo lineal. Sin embargo en realidad se dan, en ocasiones, circunstancias no-lineales.

Es fácil encontrar ejemplos sencillos de expresiones que, siendo aparentemente lineales, en el fondo no lo son, como por ejemplo el objetivo de minimizar el valor absoluto de una diferencia.

También es habitual encontrar relaciones entre variables cuya aproximación es difícilmente aproximable a un caso lineal.



Así por ejemplo los beneficios asociados a la venta de un producto pueden depender de la cantidad vendida.

Un modo bastante acertado de representar esta relación entre el precio de venta y la cantidad pedida es la denominada elasticidad de precios (la inelasticidad supone que un incremento en el precio supone incremento en los beneficios). El grado de elasticidad de un producto es:

$$E_x = \frac{\text{porcentaje de cambio en la cantidad x demandada}}{\text{porcentaje de cambio en el precio}}$$

De este modo, y asumiendo $E_x$ constante en un rango:

$$p(x) = \frac{k}{x^{1/E_x}}$$

donde *k* es el precio sobre el cual la demanda es unitaria. Por tanto los beneficios esperados para una venta de *x* son:

$$p(x) \cdot x = k \cdot x^{(1-1/E_x)}$$

***Ejemplo 3***

El ejemplo 3 es un ejemplo típico de relación no-lineal que conduciría a un objetivo no-lineal. Existen también ejemplos de restricciones no-lineales. Por ejemplo en la mezcla de componentes cuyas características mezcladas no tengan un comportamiento lineal. O cuando se vean afectadas por un producto de variables (por ejemplo, mantener la presión de un gas por debajo de ciertos niveles, pudiendo alterar volumen y temperatura).

Más habituales aún son las restricciones de tipo lógico, que por su propia estructura no son lineales.

Los modelos de programación no-lineal son mucho más complejos de resolver que sus análogos lineales, además en muchos paquetes aunque se encuentra una solución ésta no es necesariamente óptima. Sin embargo es posible convertir relaciones no-lineales en aproximaciones lineales. Y ese es el objeto de este capítulo.

## *5.1 Objetivos no-lineales fácilmente linealizables*

En los siguientes apartados se relacionan algunos de los objetivos que más comúnmente se encuentran en la construcción de modelos pero que no son lineales en su definición. Dichos objetivos son:

- Minimizar un valor absoluto
- Minimizar el máximo
- Minimizar un ratio

### 5.1.1 Objetivo "Minimizar un valor absoluto"

Esta situación se da cuando se pretende minimizar una cierta discrepancia entre dos valores.

$$[Minimize] \quad |x - y|$$

La función absoluto no es una función lineal. Para convertirla en lineal es necesario establecer una tercera variable que:

$$[Minimize] \quad z$$

Sujeto a:

$$z \geq x - y$$
$$z \geq y - x$$

### 5.1.2 Objetivo "Minimizar el Máximo" o "Maximizar el Mínimo"

En ocasiones el objetivo que se persigue se representa de la siguiente manera:

$$[\text{Minimize}]\ \underset{i}{\text{Max}}\left\{\sum_j a_j \cdot x_{ij}\right\}$$

La función máximo no es una función lineal. Y por tanto debe ser linealizada. El objetivo minimizar el máximo se puede convertir en un objetivo lineal introduciendo una variable *z* que lo representa. La nueva formulación es:

$$[\text{Minimize}]\ z$$
$$Sujeto\,a:$$
$$\sum_j a_j \cdot x_{ij} - z \le 0 \quad \forall i$$

Las nuevas restricciones garantizan que *z* será mayor o igual que cualquiera de los valores

$$\sum_j a_j \cdot x_{ij} \quad \forall i$$

Evidentemente el objetivo $[\text{Minimize}]\ \underset{i}{\text{Max}}\left\{\sum_j a_j \cdot x_{ij}\right\}$ se puede representar también de un modo similar.

Sin embargo los objetivos maximizar el máximo y minimizar el mínimo no se pueden representar de modo lineal, sino según Programación Entera (y se presentan en un apartado posterior).

### 5.1.3 Objetivos de Ratio

En ocasiones aparecen objetivos de ratio. Por ejemplo maximizar la relación beneficios/inversión. Dichos siguientes objetivos no-lineales pueden ser representados genéricamente del siguiente modo.

$$[\text{Maximize}]\ \frac{\sum_j a_j \cdot x_j}{\sum_j b_j \cdot x_j}$$

Es posible transformar la función objetivo de forma que se convierta en un objetivo lineal. Para ello hay que establecer una transformación de variables como la que se representa a continuación:

$$w_j = x_j \cdot t$$

Donde $t = \dfrac{1}{\sum_j b_j \cdot x_j}$

Con esta transformación el objetivo pasa a ser lineal, con la siguiente expresión:

$$[\text{Maximize}]\sum_j a_j \cdot w_j$$

Para garantizar la transformación hay que incorporar la siguiente restricción.

$$\sum_j b_j \cdot w_j = 1$$

Y además hay que convertir las restricciones originales que utilizan las variables *x* que han sido transformadas

$$\sum_j d_j \cdot x_j \begin{matrix} \leq \\ = \\ \geq \end{matrix} e$$

en nuevas restricciones que usan la variable $w_j$

$$\sum_j d_j \cdot w_j - e \cdot t \begin{matrix} \leq \\ = \\ \geq \end{matrix} 0$$

Hay que destacar que esta transformación es válida si $\sum_j b_j \cdot x_j$ es siempre del mismo signo y no nula.

### 5.1.4 Objetivos Maximizar el Máximo (o Minimizar el Mínimo)

Si el objetivo fuera del tipo

$$[Maximize] \operatorname{Max}_i \left\{ \sum_j a_{ij} x_j \right\}$$

*Sujeto a* *restricciones lineales convencionales*

Este tipo de situación se puede modelar del siguiente modo

$$[Maximize] \quad z$$

$$Sujeto\,a:$$

$$(\sum_j a_{1j} x_j - z = 0) \vee (\sum_j a_{2j} x_j - z = 0) \vee \ldots$$

Pero eso exige el uso de restricciones de tipo lógico que se analizarán más adelante.

## *5.2 El uso de variables discretas para representar relaciones condicionales*

El uso de variables enteras en el proceso de modelado puede servir a diferentes propósitos como se ha indicado anteriormente. Pueden representar cantidades indivisibles y también variables de decisión.

Estas indican la decisión a tomar. Suelen ser variables de tipo binario {0,1} (por ejemplo: $\delta = 1$ abrir centro y $\delta = 0$ no abrir). Aunque nada impide que adopten más valores ($\alpha = 0$ no abrir, $\alpha = 1$ abrir en Valencia, $\alpha = 2$ abrir en Alicante).

Sin embargo, en este apartado, se pretende establecer la utilidad de las variables enteras para establecer relaciones entre restricciones basadas en condiciones.

### 5.2.1 Funciones no continuas

La suposición de que la producción de un determinado artículo supone un coste fijo $C_1$ además de un coste variable $C_2$ que depende de la cantidad, y que el coste fijo no se paga si la cantidad fabricada es 0.

$$C_T = \begin{cases} C_1 + C_2 \cdot x & \text{x} > 0 \\ 0 & \text{x} = 0 \end{cases}$$

Sería interesante conocer, mediante una variable $\delta$ si $x$ es mayor que 0 o es nulo ($\delta=1$ o $\delta=0$ respectivamente). De este modo

$$C_T = C_1 \cdot \delta + C_2 \cdot x$$

Esto se podría lograr estableciendo la restricción

$$x - M \cdot \delta \leq 0$$

donde M es una cota superior conocida de $x$.

### 5.2.2 Relación Lógica $\delta = 1 \rightarrow \sum_j a_j \cdot x_j \leq b$

También se pueden utilizar estas variables indicadoras para conocer si una restricción es violada o no. Sea la restricción:

$$\sum_j a_j \cdot x_j \leq b$$

Supóngase que se desea que $\delta = 1$ represente que la restricción se cumple, esto se puede expresar como:

$$\delta = 1 \rightarrow \sum_j a_j \cdot x_j \leq b$$

Se puede mostrar que el modo de representar ésta condición mediante una desigualdad lineal es:

$$\sum_j a_j \cdot x_j + M \cdot \delta \leq M + b$$

Donde $M$ es tal que $\sum_j a_j \cdot x_j - b \leq M$ en cualquier circunstancia.

### 5.2.3 Relación Lógica $\sum_j a_j x_j < b \rightarrow \delta = 1$

Podría interesar representar la siguiente condición:

$$\sum_j a_j \cdot x_j < b \rightarrow \delta = 1$$

Esta condición se representa del siguiente modo:

$$\sum_j a_j \cdot x_j + m \cdot \delta \geq b$$

Donde *m* es una cota inferior de $\sum_j a_j \cdot x_j - b$.

### 5.2.4 Relación Lógica $\sum_j a_j x_j \leq b \rightarrow \delta = 1$

Podría interesar representar la siguiente condición

$$\sum_j a_j \cdot x_j \leq b \rightarrow \delta = 1$$

Esta condición se representa del siguiente modo:

$$\sum_j a_j \cdot x_j - \delta \cdot (m - \varepsilon) \geq b + \varepsilon$$

Donde *m* es una cota inferior de $\sum_j a_j \cdot x_j - b$ y $\varepsilon$ es la tolerancia a partir de la cual consideramos que $\sum_j a_j \cdot x_j > b$

### 5.2.5 Relación Lógica $\delta = 1 \rightarrow \sum_j a_j \cdot x_j \geq b$ :

Sea ahora la restricción:

$$\sum_j a_j \cdot x_j \geq b$$

Supóngase que se desea que *δ=1* represente que la restricción se cumple, esto se puede expresar como:

$$\delta = 1 \rightarrow \sum_j a_j \cdot x_j \geq b$$

Se puede mostrar que el modo de representar ésta condición mediante una desigualdad lineal es:

$$\sum_j a_j \cdot x_j + m \cdot \delta \geq m + b$$

Donde *m* es una cota inferior de $\sum_j a_j \cdot x_j - b$

### 5.2.6 Relación Lógica $\sum_j a_j \cdot x_j > b \rightarrow \delta = 1$

Podría interesar representar la siguiente condición

$$\sum_j a_j \cdot x_j > b \rightarrow \delta = 1$$

Esta condición se representa del siguiente modo:

$$\sum_j a_j \cdot x_j - M \cdot \delta \leq b$$

Donde *M* es de nuevo una cota superior de $\sum_j a_j \cdot x_j - b$

### 5.2.7 Relación Lógica $\sum_j a_j \cdot x_j \geq b \rightarrow \delta = 1$

Asimismo, podría interesar representar la siguiente condición

$$\sum_j a_j \cdot x_j \geq b \rightarrow \delta = 1$$

Esta condición se representa del siguiente modo:

$$\sum_j a_j \cdot x_j - \delta \cdot (M + \varepsilon) \leq b - \varepsilon$$

Donde *M* es de nuevo una cota superior de $\sum_j a_j \cdot x_j - b$ y $\varepsilon$ es la tolerancia a partir de la cual se considera que $\sum_j a_j \cdot x_j < b$

## *5.3 Más Relaciones Lógicas y su representación*

Para poder abordar este apartado es necesario establecer una cierta notación del Algebra Booleana básica:

‘$\vee$’ significa ‘O‘ (‘O INCLUSIVO’: A, B, o ambos)
‘$\wedge$’ significa ‘Y‘
‘$\neg$‘ significa ‘NO‘
‘$\rightarrow$’ significa ‘IMPLICA‘
‘$\leftrightarrow$’ significa ‘SI Y SOLO SI‘

Estos elementos se denominan operadores y unen proposiciones como P,Q,R... *x>0*, *δ=1*, etc.

El par de operadores { ¬ , ∨ } sirve para representar cualquier combinación posible (lo que es especialmente interesante). Asumiremos que los símbolos por orden de importancia son '¬' , '∧' , '∨' y '→', lo cual evita el uso de excesivos paréntesis.

Así por ejemplo:

(P∧Q)∨R se puede escribir como P∧Q∨R

P→(Q∨R) se puede escribir como P→Q∨R

¬ ¬ P es lo mismo que P

P→Q es lo mismo que ¬ P∨Q

P→Q∧R es lo mismo que (P→Q)∧(P→R)

P→Q∨R es lo mismo que (P→Q)∨(P→R)

P∧Q→R es lo mismo que (P→R)∨(Q→R)

P∨Q→R es lo mismo que (P→R)∧(Q→R)

¬ (P∨Q) es lo mismo que ¬ P∧ ¬ Q

¬ (P∧Q) es lo mismo que ¬P ∨ ¬Q

Mediante estas reglas básicas es posible construir casi cualquier expresión y reducirla a otra que utilice distintos operadores.

Así por ejemplo si $X_i$ es la proposición $\delta_i=1$, se puede decir que:

$$
\begin{aligned}
X_1 \vee X_2 &\text{ es equivalente a } \delta_1+\delta_2 \geq 1\\
X_1 \wedge X_2 &\text{ es equivalente a } \delta_1=1, \delta_2=1\\
\neg X_1 &\text{ es equivalente a } 1\text{-}\delta_1=1 \text{ ó } \delta_1=0\\
X_1 \rightarrow X_2 &\text{ es equivalente a } \delta_1-\delta_2 \leq 0\\
X_1 \leftrightarrow X_2 &\text{ es equivalente a } \delta_1-\delta_2=0
\end{aligned}
$$

Suponga que se quiere garantizar que si *x=1* entonces *y=0*, siendo ambas variables binarias

$$x+y\leq 1$$

***Ejemplo 4***

Suponga que se quiere garantizar que si $x\leq b$ y $x\geq 1$ entonces $y=z+1$. Siendo *x, y, z* variables enteras.

La expresión buscada es:

$$(x \leq b) \wedge (x \geq 1) \rightarrow (y = y+1)$$
$$(x \leq b) \wedge (x \geq 1) \rightarrow (\pi = 1) \rightarrow (y = y+1)$$

Usando:

$$(x \le b) \rightarrow (\alpha = 1)$$
$$(x \ge 1) \rightarrow (\beta = 1)$$
$$(\alpha = 1) \wedge (\beta = 1) \rightarrow (\pi = 1)$$
$$(\pi = 1) \rightarrow (y \ge z + 1)$$
$$(\pi = 1) \rightarrow (y \le z + 1)$$

Se puede afirmar que la expresión es equivalente a:

$$\alpha + \beta - \pi \le 1$$
$$x + M \cdot \alpha \ge b + 1$$
$$x - M \cdot \beta \le 0$$
$$y - z - m \cdot \pi \ge m + 1$$
$$y - z + M \cdot \pi \le M + 1$$

Donde m es una cota inferior de y-z-1 y M es una cota superior de y-z+1

***Ejemplo 5***

En un caso de programación de producción, si cualquiera de los productos A o B se fabrican, entonces hay que fabricar C, D o E.

Esta condición se puede representar como

$$(X_A \vee X_B) \rightarrow (X_C \vee X_D \vee X_E)$$

Sea $\delta_i=1 \leftrightarrow$ se fabrica el producto i

Sea $\delta$ una variable auxiliar tal que

$$\delta_A + \delta_B \ge 1 \rightarrow \delta = 1$$

Esta expresión se puede indicar como

$$\delta_A + \delta_B + 2 \cdot \delta \le 0$$

Además

$$\delta = 1 \rightarrow \delta_C + \delta_D + \delta_E \ge 1$$

Lo que se puede expresar como

$$\delta - \delta_C - \delta_D - \delta_E \le 0$$

Así pues el ejemplo se puede representar como el par de restricciones

$$\left.\begin{array}{l}\delta_A + \delta_B - 2\delta \le 0 \\ \delta - \delta_C - \delta_D - \delta_E \le 0\end{array}\right\}$$

***Ejemplo 6***

En ocasiones se admite como útil el uso de polinomios de variables indicadoras. Estas expresiones pueden ser siempre sustituidas por expresiones lineales, así, por ejemplo

$$\delta_1 \cdot \delta_2 = 0 \text{ es equivalente a } \delta_1 + \delta_2 \le 1$$
$$\delta_1 \cdot \delta_2 = 1 \text{ es equivalente a } \delta_1 + \delta_2 = 2$$

De modo más general si el término $\delta_1 \cdot \delta_2$ ha de aparecer siempre podrá ser sustituido por una variable $\delta_3 = \delta_1 \cdot \delta_2$ que tiene la siguiente condición $\delta_3 = 1 \leftrightarrow (\delta_1 = 1 \wedge \delta_2 = 1)$ lo cual se puede expresar mediante

$$\begin{array}{l}-\delta_1 + \delta_3 \le 0 \\ -\delta_2 + \delta_3 \le 0 \\ \delta_1 + \delta_2 - \delta_3 \le 1\end{array}$$

Es incluso posible linealizar términos en los que se multiplica el valor de una variable 0-1 con una variable real $(x \cdot \delta)$. Sea $y = (x \cdot \delta)$, tenemos

$$\left.\begin{array}{l}\delta = 0 \rightarrow y = 0 \\ \delta = 1 \rightarrow y = x\end{array}\right\}$$

lo que es equivalente a

$$\begin{array}{rl} y - M \cdot \delta & \le 0 \\ -x + y & \le 0 \\ x - y + M \cdot \delta & \le M \end{array}$$

## *5.4 Conjuntos especiales de variables ordenadas*

Existen dos tipos de restricciones cuya presencia en problemas reales es habitual y que, además, están implementadas en múltiples paquetes porque facilitan la resolución. Estas son conjuntos de variables en que sólo una de entre todas puede adoptar un valor no nulo. Estas variables se denominan SOS1.

***Ejemplo 7***

El modelado de una función como

$$y = \begin{cases} 2x & \text{x} \in [0,4] \\ x+4 & \text{x} \in [4,6] \\ 3x-8 & \text{x} \in [6,10] \end{cases}$$

sería del tipo:

$$x = 4\lambda_2 + 6\lambda_3 + 10\lambda_4$$
$$y = 8\lambda_2 + 10\lambda_3 + 22\lambda_4$$
$$\lambda_1 + \lambda_2 + \lambda_3 + \lambda_4 = 1$$

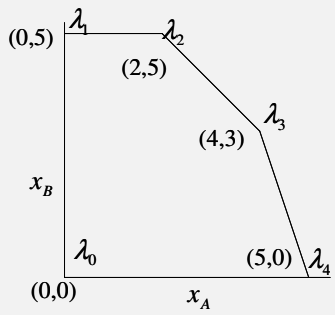
siendo ( $\lambda_1, \lambda_2, \lambda_3, \lambda_4$ )un conjunto SOS1

Otro ejemplo de uso de estas variables es la denominada formulación modal, citada en un apartado anterior.

Sea el siguiente conjunto de restricciones:

$$x_A + x_B \leq 7$$
$$3 \cdot x_A + x_B \leq 15$$
$$x_B \leq 5$$

La representación gráfica del espacio de soluciones de este modelo es:




Esta situación se puede modelar representando exclusivamente los vértices. Usando por tanto más variables aunque menos restricciones (concretamente una), tenemos:

$$\lambda_0 + \lambda_1 + \lambda_2 + \lambda_3 + \lambda_4 = 1$$

Y allí donde se encuentra $x_A$ y $x_B$ los cambiaremos por:

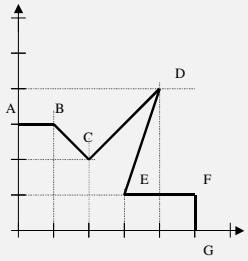
$$x_A = 2\lambda_2 + 4\lambda_3 + 5\lambda_4$$
$$x_B = 5\lambda_1 + 5\lambda_2 + 3\lambda_3$$

Donde los coeficientes son las coordenadas *x* e *y* de cada uno de los puntos. Este tipo de formulación puede ahorrar gran cantidad de tiempo en la resolución del problema.

Asociadas a estas existen los conjuntos de variables denominadas SOS2. En ellos sólo dos variables pueden tener valor no-nulo y además deben ser consecutivas.

***Ejemplo 8***

Supóngase que la región de factibilidad de una variable es la siguiente OABCDEFG región no-convexa




Esta región se puede representar de la siguiente manera, utilizando las variables $\delta_1, \delta_2 y \delta_3$

$$\delta_1 = 1 \rightarrow (x_2 \leq 3) \wedge (x_1 + x_2 \leq 4)$$
$$\delta_2 = 1 \rightarrow (-x_1 + x_2 \leq 0) \wedge (3x_1 - x_2 \leq 8)$$
$$\delta_3 = 1 \rightarrow (x_2 \leq 1) \wedge (x_1 \leq 5)$$

lo que equivale a imponer las siguientes restricciones:

$$x_2 + \delta_1 \leq 4$$
$$x_1 + x_2 + 5\delta_1 \leq 9$$
$$-x_1 + x_2 + 4\delta_2 \leq 4$$
$$3x_1 - x_2 + 7\delta_2 \leq 15$$
$$x_2 + 3\delta_3 \leq 4$$
$$x_1 \leq 5$$
$$\delta_1 + \delta_2 + \delta_3 \geq 1$$

Existe una formulación alternativa para una región no-convexa y conectada. Esta sería la siguiente

$$x_1 - \lambda_B - 2\lambda_C - 4\lambda_D - 3\lambda_E - 5\lambda_F - 5\lambda_G = 0$$
$$x_2 - 3\lambda_A - 3\lambda_b - 2\lambda_C - 4\lambda_D - \lambda_E - \lambda_F = 0$$
$$\lambda_A + \lambda_B + \lambda_C + \lambda_D + \lambda_E + \lambda_F \leq 1$$

Siendo en este caso el conjunto $(\lambda_A, \lambda_B, \lambda_C, \lambda_D, \lambda_E, \lambda_F)$ del tipo SOS2

El uso de estos dos tipos de variables SOS1 y SOS2 es considerado explícitamente por algunos motores de resolución y facilita enormemente el proceso de ramificación y corte, y por tanto reduce el coste computacional.

# 6 MODELOS DE PROGRAMACIÓN ENTERA

## 6.1 Introducción

Una gran cantidad de problemas reales se puede modelar utilizando variables enteras y restricciones lineales.

En la mayor parte estos los valores que pueden adoptar las variables enteras son 0-1. Tales valores se utilizan para modelar decisiones "si o no". Como se verá en los siguientes apartados, es posible establecer relaciones lógicas entre restricciones, usando este tipo de variables y manteniendo la linealidad.

Antes de abordar la linealización mediante programación entera, hay que destacar la problemática de la resolución de estos modelos. Los tiempos de computación para resolver un problema de Programación Entera son mucho más largos que para un problema de Programación Lineal de similares características.

Existe un peligro evidente de invertir gran cantidad de recursos en modelar, sólo para descubrir al final que el problema no era resoluble en un tiempo razonable, por ello en los últimos apartados del presente capítulo se abordará el problema de reducir el tamaño de los modelos.

## 6.2 Diferentes áreas de aplicación de la Programación Entera

En el presente apartado se tratan diferentes áreas donde la programación entera resulta de interés.

### 6.2.1 Problemas con inputs (u outputs) discretos

Este es el caso más obvio: si fabricamos aviones o coches no es posible fabricar 2.33 unidades. Por tanto la solución debe ser entera. En realidad en estos casos es especialmente relevante la PE sólo si los valores resultado son de un dígito. En otros casos basta resolver como si fuera Programación Lineal y posteriormente redondear.

### 6.2.2 Problemas con condiciones lógicas

Suele ocurrir que en algunas de las restricciones que se deseen imponer son de tipo lógico.

*"si se abre fábrica en Valencia, no se debe abrir en Madrid"*

Este tipo de condiciones lógicas pueden ser modeladas mediante Programación Entera, simplemente introduciendo nuevas variables como se verá en un apartado posterior.

### 6.2.3 Problemas de combinatoria

Muchos de los problemas prácticos de Investigación Operativa tienen como característica básica la existencia de un extremadamente largo número de soluciones posibles. Dichas soluciones aparecen a partir de diferentes métodos de ordenar actividades y/o asignar recursos.

Este tipo de problemas se denominan “Combinatorios”. Los problemas de tipo combinatorio se pueden clasificar a su vez en problemas de “Secuencia” o de “Reparto”. Estos últimos se clasifican a su vez en problemas de asignación, donde existen tantos trabajos como recursos, y cada trabajo exige sólo un recurso, y los problemas de transporte donde cada trabajo requiere varios recursos. Algunos problemas de Reparto no requieren PE, aunque la mayoría sí.

### 6.2.4 Problemas No-lineales

Como ya se ha comentado en el capítulo anterior los problemas No-Lineales se pueden atacar mediante modelos de PE. Para ello se han mostrado diferentes técnicas en el capítulo anterior.

## *6.3 Otras condiciones aplicadas a Modelos de Programación Lineal*

La mayor parte de los modelos de Programación Entera Mixta (PEM) aparecen desde Programación Lineal en la que hay que aplicar unas restricciones extraordinarias. En este apartado se observan las más comunes de estas restricciones.

### 6.3.1 Restricciones disyuntivas

Supongamos que en un problema de Programación Lineal pretendemos que no todas las restricciones actúen simultáneamente

$$R_1 \vee R_2 \vee R_3 \vee ... \vee R_N$$

Introduciremos un conjunto de variables $\delta_i$ (i=1,...,N) donde $\delta_i = 1$ si se ha de cumplir $R_i$.

$$\delta_i = 1 \rightarrow R_i$$

Y el conjunto $\delta_i$ debe cumplir

$$\delta_1 + \delta_2 + ... + \delta_N \geq 1$$

***Ejemplo 9***

Sea una restricción $R_1 \vee R_2 \vee R_3$

Si $R_i$ es del tipo $\sum_j a_{ij} \cdot x_j \leq b_i$, entonces el caso se representa como:

$$\sum_j a_{1j} \cdot x_j + M \cdot \delta_1 \leq M + b_1$$
$$\sum_j a_{2j} \cdot x_j + M \cdot \delta_2 \leq M + b_2$$
$$\sum_j a_{3j} \cdot x_j + M \cdot \delta_3 \leq M + b_3$$
$$\delta_1 + \delta_2 + \delta_3 \geq 1$$

### 6.3.2 Regiones No-Convexas

### 6.3.3 Limitar el número de variables en una solución

Esta es otra aplicación de restricciones disyuntivas. Como ya se ha comentado en cualquier problema de Programación Lineal no habrá más variables no-nulas que restricciones. Sin embargo es posible que nos interese reducir este número de variables no-nulas. Para hacer esto se necesita la PE.

Para ello se incorporan nuevas variables $\delta_i$ tales que:

$$x_i \geq 0 \rightarrow \delta_i = 1$$

lo que equivale a introducir la restricción $x_i - M \cdot \delta_i \leq 0$

En este caso se introducirá la siguiente condición

$$\sum_i \delta_i \leq k$$

que reducirá a k el número de variables no-nulas

### 6.3.4 Decisiones Secuencialmente Dependientes

A menudo ocurre que se pretende que una decisión se prolongue en el tiempo. Por ejemplo si $\gamma_i$ representa si una instalación está abierta o cerrada, y se pretende que una vez cerrada ya no se vuelva a abrir

$$\gamma_i = 0 \rightarrow (\gamma_{i+1} = 0) \wedge (\gamma_{i+2} = 0) \wedge ... \wedge (\gamma_n = 0)$$

se puede representar como

$$\begin{aligned} -2\gamma_1 + \gamma_2 &\leq 0 \\ -2\gamma_2 + \gamma_3 &\leq 0 \\ &.... \\ -2\gamma_{n-1} + \gamma_n &\leq 0 \end{aligned}$$

### 6.3.5 Extensiones discretas de capacidad

A menudo no es realista suponer que determinadas restricciones no pueden ser violadas. Así por ejemplo una máquina podría aumentar su número de horas disponibles sin más que duplicar (o triplicar) el número de turnos que trabaja. Con evidentemente un coste asociado.

Supongamos que la restricción citada se puede expresar como

$$\sum_i a_i \cdot x_i \leq b_j$$

Donde $b_j$ es variable y tiene un coste asociado de $c_j$ (independiente para cada $b_j$) con $0 < c_o < c_1 < ... < c_n$

Esta situación se puede modelar del siguiente modo en la restricción

$$\sum_i a_i \cdot x_i - b_o\delta_0 - b_1\delta_1 - ... - b_n\delta_n \leq 0$$

y en la función objetivo se añade

$$c_o\delta_0 + c_1\delta_1 + ... + c_n\delta_n$$

el conjunto de variables $(\delta_0, \delta_1, ..., \delta_n)$ es de tipo SOS1 y fijándole una cota de 1 no es siquiera necesario establecer condiciones de integralidad.

## *6.4 Tipos Especiales de Modelos de Programación Entera*

En estos apartados se presentan algunos problemas clásicos de Programación Entera. La importancia de estos problemas radica en el hecho de que los problemas complejos se suelen poder reducir a combinaciones de éstos.

### 6.4.1 El problema de la mochila

Un problema de PE con una única restricción se denomina “Problema de Mochila” y toma la forma

$$[Maximize] \quad \sum_i c_i \cdot \delta_i$$
$$Sujeto\ a:$$
$$\sum_i a_i \cdot \delta_i \leq b$$

El problema de la mochila se da pocas veces de modo independiente (selección de proyectos o decisión de inversión). El problema de la mochila es, comparativamente, fácil de resolver. El mejor modo de resolverlo es programación dinámica, y no ramificación y corte que explora de manera demasiado amplia.

### 6.4.2 Problemas de cubrimiento

Estos problemas derivan de un tipo de problema abstracto que puede ser enunciado del siguiente modo:

***Ejemplo 10***

"Dado un conjunto S de objetos que podemos numerar *S=(1,2,...,m)* y dada una clase T de subconjuntos de S, teniendo cada clase un coste el problema es cubrir todos los miembros de S utilizando miembros de T".

Sea S=(1,2,3,4,5,6) sea T=((1,2),(1,3,4),(5,6),(4,6),(3,4,5)). Se produciría cubrimiento con ((1,2),(1,3,4),(5,6)) o con ((1,2),(5,6),(3,4,5)).

Para modelarlo sea:

$$\delta_i = \begin{cases} 1 & \text{si el i - ésimo elemento de T es seleccionado} \\ 0 & \text{en caso contrario} \end{cases}$$

$$c_i = \text{el coste de usar el i - ésimo elemento de T}$$

El modelo resulta

$$[Minimize] \quad \sum_i c_i \cdot \delta_i$$

$$Sujeto\ a:$$

$$\begin{aligned} \delta_1 + \delta_2 &\geq 1 \\ \delta_1 &\geq 1 \\ \delta_2 + \delta_5 &\geq 1 \\ \delta_2 + \delta_4 + \delta_5 &\geq 1 \\ \delta_3 + \delta_5 &\geq 1 \\ \delta_3 + \delta_4 &\geq 1 \end{aligned}$$

Este problema de cubrimiento tiene 3 propiedades básicas.

Propiedad 1: El problema es de minimización y las restricciones son ≥1

Propiedad 2: Todos los coeficientes RHS son 1

Propiedad 3: Todos los demás coeficientes de la matriz son 0 o 1

Si se relaja la propiedad 2 (se obliga a que algunos elementos de S sean cubiertos en un mayor número de ocasiones) se tiene un tipo de problema de cubrimiento ponderado. Y si además se permite que los demás coeficientes de la matriz puedan valer 0 y ±1 (propiedad 3) tenemos lo que se denomina un problema de cubrimiento generalizado.

### 6.4.3 Problemas de empaquetado

Otro tipo clásico de problemas son los de empaquetado, que se pueden enunciar así:

***Ejemplo 11***

"Dado un conjunto S de objetos que podemos enumerar *S=(1,2,...,m)* y dada una clase T de subconjuntos de S, teniendo cada subconjunto un valor asociado, se trata de relacionar miembros de T que maximicen el valor pero que no se superpongan".

Con el conjunto S y la clase T del ejemplo anterior el problema queda (incorporando un valor $v_i$ a cada miembro de T).

$$[Maximize] \quad \sum_i v_i \cdot \delta_i$$

$$Sujeto\ a:$$

$$\begin{aligned}
\delta_1 + \delta_2 &\le 1 \\
\delta_1 &\le 1 \\
\delta_2 + \delta_5 &\le 1 \\
\delta_2 + \delta_4 + \delta_5 &\le 1 \\
\delta_3 + \delta_5 &\le 1 \\
\delta_3 + \delta_4 &\le 1
\end{aligned}$$

Las mismas generalizaciones de los problemas de cubrimiento se pueden aplicar en los problemas de empaquetado dando lugar al problema de empaquetado ponderado y al problema de empaquetado generalizado.

### 6.4.4 El Problema del viajante de comercio

Éste problema es un problema muy habitual y de simple formulación aunque de resolución muy complicada en circunstancias reales.

El problema consiste en establecer un circuito que recorra todos los destinos pasando una solo vez por cada uno de ellos. Existen muchas situaciones reales asociadas a este problema de rutas, con diferentes variantes, e incluso se da en otras situaciones como por ejemplo en la reducción de costes de cambio de partida.

Su formulación evidente es la siguiente

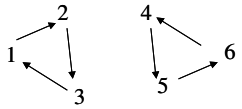
$$[Minimize] \quad \sum_i \sum_j c_{ij} \cdot \delta_{ij}$$

$$Sujeto\ a:$$

$$\sum_i \delta_{ij} = 1 \quad \forall j$$

$$\sum_j \delta_{ij} = 1 \quad \forall i$$

En una situación cualquiera (con 6 ciudades) la solución óptima podría ser:




lo que, evidentemente, no es una solución satisfactoria. Para eliminar esta circunstancia se puede añadir el siguiente conjunto de restricciones, que requieren de las variables $u_i$.

$$u_i - u_j + n \cdot \delta_{ij} \le n - 1$$

Esto incorpora n·(n-1) restricciones al problema. La variable $u_i$ representa el orden en el que el punto i será visitado, siendo $u_1=1$.

Si el número de restricciones incorporado es excesivo (lo que puede ocurrir) existe un procedimiento alternativo para resolver el problema de las sub-ciclos. Consistiría en incorporar la restricción

$$\delta_{12} + \delta_{23} + \delta_{31} \leq 2$$

Esto eliminaría la solución planteada anteriormente de entre las posibles. En definitiva éste es un modo de buscar el óptimo mediante el sucesivo refinamiento del modelo.

En cualquier caso hay que destacar el elevadísimo coste computacional de la resolución de este problema mediante PEM, cuando las dimensiones no son pequeñas por ello lo habitual es plantear mecanismos de resolución especiales.

### 6.4.5 El problema de asignación cuadrática

Esta es una generalización del problema de asignación descrita en un capítulo anterior. Aunque el problema de asignación es un problema de Programación Lineal fácil de resolver, el problema de asignación cuadrática es un problema de PE y generalmente muy difícil de resolver.

El problema consiste en:

"Dado un conjunto S con n elementos y un conjunto T con el mismo número de elementos, el problema consiste en asignar exactamente un elemento de S a T y viceversa"

Para ello se define

$$\delta_{ij} = \begin{cases} 1 & \text{si } \mathrm{i} \in \mathrm{S} \text{ se asigna a } \mathrm{j} \in \mathrm{T} \\ 0 & \text{en caso contrario} \end{cases}$$

Las restricciones son las siguientes:

$$\sum_{j=1}^{n} \delta_{ij} = 1 \qquad \mathrm{i} = 1,2,...,\mathrm{n}$$

$$\sum_{i=1}^{n} \delta_{ij} = 1 \qquad \mathrm{j} = 1,2,...,\mathrm{n}$$

La diferencia con el problema de asignación normal radica en la función objetivo. En ella las variables se multiplican, para valorar la asignación combinada. Es decir, se asigna un coste $c_{ijkl}$ si al mismo tiempo que $i \in S$ se asigna a $j \in T$ se asigna a $k \in S$ a $l \in T$.

$$\sum_{\substack{i,j,l \\ k>i}}^{n} c_{ijkl} \cdot \delta_{ij} \cdot \delta_{kl}$$

Este problema se puede abordar como un problema de PE clásico con $n^4$ variables binarias. También existen procedimientos específicos de resolución.

## *6.5 Buenas y malas formulaciones de un modelo de PE*

La mayor parte de las consideraciones realizadas en apartados anteriores para la Programación Lineal sirven para la PE. Además nuevas consideraciones se convierten en importantes en el momento que el problema tiene variables enteras.

La primera y más importante de las razones es que la resolución de un modelo de PE es mucho más complicada que uno de Programación Lineal de parecidas dimensiones.

El número de variables no es un buen indicador para saber si un modelo está bien o mal construido. De hecho en ocasiones es interesante introducir nuevas variables siempre que faciliten el proceso de resolución.

El número de restricciones afecta, generalmente, en un sentido contrario que a los modelos de Programación Lineal, es decir, un mayor número de restricciones facilita el proceso de resolución. El motivo es que aumenta la posibilidad de que el óptimo del modelo de Programación Lineal asociado sea a su vez entero.

El uso de los conjuntos ordenados de variables, si el paquete puede trabajar con ellos, facilita la resolución.

## *6.6 Simplificación de un modelo de Programación Entera*

En este apartado se proponen cuatro procedimientos que permiten mejorar el tiempo de resolución de modelos de PE. La mayor parte de estos procedimientos parten del análisis concreto de cada problema.

### 6.6.1 Más Restricciones y Más Ajustadas

En Programación Lineal es habitual pretender reducir el número de restricciones como un modo de facilitar la resolución. En PE el efecto deseado es el mismo, aunque el método es el contrario. Mediante el análisis directo de diferentes restricciones es posible encontrar restricciones combinadas que reducen el ámbito de variación de diferentes variables, haciendo que el óptimo del problema de Programación Lineal esté muy cerca del óptimo del PE.

***Ejemplo 12***

$$[Minimize] \quad 5\delta_1 + 7\delta_2 + 10\delta_3 + 3\delta_4 + \delta_5$$

$$Sujeto\ a:$$

$$\delta_1 - 3\delta_2 + 5\delta_3 + \delta_4 - \delta_5 \geq 2$$
$$-2\delta_1 + 6\delta_2 - 3\delta_3 - 2\delta_4 + 2\delta_5 \geq 0$$
$$-\delta_2 + 2\delta_3 - 2\delta_4 - 2\delta_5 \geq 1$$
$$\delta_i \in \{0,1\} \quad \forall i$$

De la tercera restricción se puede sacar la siguiente relación:

$$2\delta_3 \geq 1 + \delta_2 + 2\delta_4 + 2\delta_5 \geq 1 \quad \rightarrow \quad \delta_3 \geq 1/2 \rightarrow \delta_3 = 1$$

De la segunda restricción, y realizando operaciones parecidas:

$$6\delta_2 \geq 2\delta_1 + 3 + 2\delta_4 - 2\delta_5 \geq 1 \quad \rightarrow \quad \delta_2 \geq 1/6 \rightarrow \delta_2 = 1$$

Y nuevamente de la restricción tercera:

$$\delta_4 \leq -\delta_5 \leq 0 \quad \rightarrow \quad \begin{matrix} \delta_4 \leq 0 \rightarrow \delta_4 = 0 \\ \delta_5 = 0 \end{matrix}$$

De donde $\delta_1 = 0$ y la función objetivo es 0.

Básicamente se puede admitir que el método consiste en analizar la configuración del problema, para encontrar relaciones, más o menos evidentes, que reducen el número de variables al asignarles valores.

### 6.6.2 Simplificar una restricción entera con otra restricción

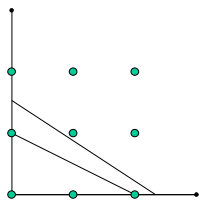
Dada la restricción entera

$$4\gamma_1 + 6\gamma_2 \leq 9$$

Donde $\gamma_1$ y $\gamma_2$ son variables enteras. A efectos de la resolución la restricción, la restricción

$$\gamma_1 + 2\gamma_2 \leq 2$$

es equivalente y mucho más adecuada porque los enteros más próximos están en ella considerados



El procedimiento consiste en encontrar para cada restricción

$$a_1 \cdot \delta_1 + a_2 \cdot \delta_2 + ... + a_n \cdot \delta_n \leq a_o$$

Una restricción

$$b_1 \cdot \delta_1 + b_2 \cdot \delta_2 + ... + b_n \cdot \delta_n \leq b_o$$

que no reduzca el espacio de enteros posibles pero sí el del problema de Programación Lineal asociado. Aunque es un método muy efectivo, realmente el cálculo de las restricciones citadas a menudo consume excesivos recursos.

### 6.6.3 Restricciones discontinuas

En ocasiones hay que re*string*ir una variable continua a segmentos o puntos, por ejemplo

$$x = 0 \quad \text{ó} \quad \mathrm{a} \leq \mathrm{x} \leq \mathrm{b} \quad \text{ó} \quad \mathrm{x} = \mathrm{c}$$

Esta circunstancia puede ser representada por un conjunto de restricciones disyuntivas, pero también existe un procedimiento alternativo para representarla

$$\begin{aligned} x &= a \cdot y_1 + b \cdot y_2 + c \cdot \delta_2 \\ \delta_1 &+ y_1 + y_2 + \delta_2 = 1 \end{aligned}$$

donde $\delta_1$ y $\delta_2$ son variables binarias

donde '$y_1$' & '$y_2$' son variables continuas (y no negativas)

Un caso común pero especial es:

$$x = 0 \,\text{ó}\, \mathrm{x} \geq \mathrm{a} \quad (\mathrm{a} > 0)$$

En este caso

$$\begin{aligned} x &= a \cdot y_1 + M \cdot y_2 \\ \delta &+ y_1 + y_2 = 1 \end{aligned}$$

## *6.7 Información económica y sensibilidad en los modelos de PE*

Por las propias restricciones de los modelos de PE no es posible extraer de la resolución (o del dual) información estricta sobre los precios sombra o los costes reducidos, de las restricciones o de los coeficientes.

Las restricciones no completamente utilizadas no implican, necesariamente que no tengan precio sombra, puesto que es posible que una ligera ampliación introduzca (o saque) de la región de factibilidad un punto que cumpla las condiciones de integralidad y mejore la función objetivo.

El único modo de conocer tanto la información económica prevista como de hacer un análisis de sensibilidad adecuado consistiría en resolver una y otra vez el mismo modelo cambiando coeficientes.

# 7 PROGRAMACIÓN NO-LINEAL

Los modelos de programación no-lineal son mucho más complejos de resolver que sus análogos lineales. Además en muchos paquetes aunque se encuentra una solución ésta no es necesariamente óptima. Sin embargo es posible convertir relaciones no-lineales en aproximaciones lineales. Para poder tomar la decisión acerca de qué método de resolución es más apropiado es importante conocer si el problema es convexo o no-convexo.

## 7.1 Óptimos locales y globales

El problema más grave de la denominada Programación No-lineal es la dificultad de garantizar que un óptimo local (es decir un punto que es objetivamente mejor que todos sus vecinos) sea el óptimo global. Mucha investigación se dedica a este problema, y estos apuntes no pretenden ni de lejos reflejarla.

Pero sí que hay que entender determinados conceptos que ayudan a comprender la complejidad del problema. Para ello es necesario recordar tres definiciones.

Definición de región convexa: Se dice que una región del espacio es convexa si el segmento que une dos puntos cualesquiera de la región está íntegramente en la región.

Definición de función convexa: Se dice que una función es convexa si el conjunto de los puntos $(x,y)$ donde $y \geq f(x)$ forma un región convexa.

Definición de un modelo de PM convexo: Se dice que un modelo de Programación Matemática es convexo si implica la minimización de una función convexa sobre una región convexa.

***Ejemplo 13***

*Un modelo de Programación Matemática Convexo:*

$$\begin{aligned} &[Minimize] \quad x_1^2 - 4x_1 - 2x_2 \\ &Sujeto\,a: \\ &x_1 + x_2 \leq 4 \\ &2x_1 + x_2 \leq 5 \\ &-x_1 + 4x_2 \geq 2 \\ &x_1, x_2 \geq 0 \end{aligned}$$

***Ejemplo 14***

*Un modelo de Programación Matemática No-Convexo*

$$
\begin{aligned}
&[Minimize] \quad -4x_1^3 + 3x_1 - 6x_2 \\
&Sujeto\,a: \\
&\quad x_1 + x_2 \le 4 \\
&\quad 2x_1 + x_2 \le 5 \\
&\quad -x_1 + 4x_2 \ge 2 \\
&\quad x_1, x_2 \ge 0
\end{aligned}
\tag{1}
$$

En el ejemplo 14 la región de factibilidad es convexa, aunque la función objetivo no lo es, por lo que es modelo no es convexo. La no-convexidad conduce a una situación en la que aparecen óptimos locales lejos del óptimo global.

La posibilidad de que óptimos locales aparezcan como óptimos propiamente dichos, cuando el modelo es no-convexo, es lo que convierte estos problemas en muy difíciles de resolver.

Si el problema es convexo cualquier óptimo que se encuentre es óptimo global. Sin embargo encontrar un óptimo global en un problema no-convexo requiere algoritmos más sofisticados. Por ejemplo el uso de la programación separable que se explica en la siguiente sección.

## *7.2 Programación Separable*

Se dice que una función es separable si puede ser expresada como la suma de funciones de una variable única.

Por ejemplo, la función

$$x_1^3 + \frac{1}{x_2} + \log(x_3) \tag{2}$$

se puede denominar separable mientras que

$$x_1 \cdot x_2 + \frac{x_3}{x_1 + x_2} \tag{3}$$

no es función separable.

La importancia de que una función sea separable para la programación matemática radica en el hecho de que una vez separada la mayor parte de las funciones se pueden aproximar a funciones definidas por tramos que sean lineales. A partir de aquí será posible encontrar un óptimo global para una función convexa, o al menos un óptimo local si es no-convexa.

Aunque puede parecer que la clase de funciones separables es muy restrictiva, en realidad es posible convertir en separables muchas funciones objetivo aparentemente no separables.

Una vez las variables han sido separadas, la función resultante se puede aproximar a una función por tramos lineal. No importa donde esté la no-linealidad, pues el procedimiento es el mismo tanto se es en la función objetivo como si es en las restricciones.

***Ejemplo 15***

Para convertir en lineal la siguiente función objetivo del Ejemplo 13:

$$x_1^2 - 4x_1 - 2x_2$$

solo es necesario convertir la función $x_1^2$.

A partir de la segunda restricción del ejemplo es evidente que no es posible que $x_1$ sea mayor que 2.5.

En este caso La función

$$y = x_1^2 \qquad x \in [0,2.5]$$

puede ser sustituida por:

$$y = \begin{cases} x & \text{x} \in [0,1] \\ 3x - 2 & \text{x} \in [0,2] \\ \dfrac{2.25}{0.5}x - 5 & \text{x} \in [2,2.5] \end{cases}$$

Esta representación es equivalente a:

$$\begin{aligned} x_1 &= 0\lambda_1 + 1\lambda_2 + 2\lambda_3 + 2.5\lambda_4 \\ y &= 0\lambda_1 + 1\lambda_2 + 4\lambda_3 + 6.25\lambda_4 \\ \lambda_1 &+ \lambda_2 + \lambda_3 + \lambda_4 = 1 \end{aligned}$$

Por tanto el modelo del ejemplo 1 es el siguiente:

$$\begin{aligned} [Minimize] \quad & y - 4x_1 - 2x_2 \\ Sujeto\,a: \quad & \\ & x_1 + x_2 \le 4 \\ & 2x_1 + x_2 \le 5 \\ & -x_1 + 4x_2 \ge 2 \\ & -x_1 + \lambda_2 + 2\lambda_3 + 2.5\lambda_4 = 0 \\ & -y + \lambda_2 + 4\lambda_3 + 6.25\lambda_4 = 0 \\ & \lambda_1 + \lambda_2 + \lambda_3 + \lambda_4 = 1 \\ & y, x_1, x_2, \lambda_1, \lambda_2, \lambda_3, \lambda_4 \ge 0 \end{aligned}$$

Dado que la función es convexa no es necesario añadir ninguna restricción más. Si la función fuera no-convexa sería necesario añadir una restricción que tuviera en cuenta que

*"como máximo 2 $\lambda_i$ adyacentes pueden ser no nulos"*

Esta restricción sólo se puede modelar mediante Programación Entera, como se vio en un apartado anterior. Lo destacable es que existen implementaciones que la consideran de manera eficiente.

En cualquier caso la separación de variables para generar problemas no convexos conduce con demasiada frecuencia a óptimos locales.

Para resolver este problema a menudo es posible resolver el mismo modelo mediante diferentes estrategias, lo que da lugar a diferentes óptimos locales y, con suerte, algunos de ellos es óptimo global.

Existe un modo más eficiente de convertir una función no-lineal en una función lineal. Este modo alternativo conduce a la siguiente formulación del Ejemplo 13.

$$x_1 = \delta_1 + \delta_2 + 0.5\delta_3$$
$$y = \delta_1 + 3\delta_2 + 2.25\delta_3$$

$$\text{“}\delta_i > 0 \rightarrow \left(\delta_j = 1 \ \forall \mathrm{j} < \mathrm{i}\right) \wedge \left(\delta_j > 0 \ \forall \mathrm{j} > \mathrm{i}\right)\text{”}$$

Aunque en tiempo de computación es más eficiente, este sistema es más difícil de modelar y requiere necesariamente del uso de programación entera. Y en cualquier caso el problema de óptimos locales permanece.

## *7.3 Cómo convertir un modelo no-separable en un modelo separable*

Considerar la no-linealidad sólo si el modelo es separable puede parecer demasiado restrictivo. Sin embargo, es posible convertir casi cualquier función de dos o más variables en una función separable.

Una de las funciones no-separables más comunes es el producto de dos o más variables. Si observamos en nuestra función la expresión:

$$x_1 \cdot x_2$$

ésta puede ser sustituida por $u_1^2 - u_2^2$ donde

$$u_1 = \frac{1}{2}(x_1 + x_2)$$
$$u_2 = \frac{1}{2}(x_1 - x_2)$$

que es una función separable, donde los términos pueden ser linealizados de modo aproximado como se explica en el apartado anterior. Hay que destacar que $u_2$ es un variable libre, es decir, puede adoptar valores negativos.

Otro método de separación es el uso de logaritmos. La anterior función $x_1 \cdot x_2$ puede ser representada por:

$$'y' \quad donde \quad \log(y) = \log(x_1) + \log(x_2)$$

que es una restricción que puede tener variables separadas. Y donde la función logaritmo puede ser linealizada.

Un tercer modo de separación es el uso de funciones lineales definidas por tramos de más de una dimensión. Aunque este modo obliga a la aparición de un gran número de variables y restricciones auxiliares.

# 8 PROGRAMACIÓN ESTOCÁSTICA

## 8.1 Introducción

Hasta este momento todas las formulaciones que se han presentado de Programación Matemática asumen que los datos son conocidos, ciertos y exactos. Sin embargo, en muchos problemas reales los datos no pueden ser conocidos con exactitud. En ocasiones por errores de medida, pero más generalmente porque son datos sobre circunstancias que se darán en el futuro, y simplemente no pueden ser conocidos con anticipación.

Este tipo de problemas se dan con mucha frecuencia, puesto que en muchas ocasiones las decisiones se toman de modo recurrente.

Por ejemplo el departamento de compras de una empresa toma la decisión de adquirir cierta cantidad de materia prima atendiendo a una demanda conocida para el presente pero estimada para el futuro. Y cuando ese futuro sea presente se tomará una nueva decisión.

El anterior es un ejemplo de recursividad. Ésta es, básicamente, la capacidad de emprender acciones correctivas después de que la situación incierta ocurra.

Hay que destacar también la propiedad de no-anticipación que se exige a la programación estocástica. Se puede definir esta como la imposibilidad de vincular la decisión en el primer periodo no depende de lo que de hecho ocurra en el segundo periodo. *La decisión de hoy para hoy no puede ser modificada mañana.*

## 8.2 Formulación de Problema Estocástico

Una posible formulación del Problema Lineal Estocástico es la siguiente:

$$[Minimize] \quad c^T \cdot x + \sum_e p_e \cdot d_e^T \cdot y_e$$

$$Sujeto\ a:$$

$$A \cdot x \le b$$
$$T_e \cdot x + W_e \cdot y_e \le h_e \quad \forall e$$
$$x \ge 0$$
$$y_e \ge 0 \quad \forall e$$

Donde:

- $c^T \cdot x$ : Coste de la decisión inmediata
- $d_e^T \cdot y_e$ : Coste de la mejor decisión en el escenario *e*
- $p_e$ : Probabilidad del escenario e
- $A$ : Matriz tecnológica sin incertidumbre
- $T_e$ : Matriz tecnológica con incertidumbre para las variables que corresponden a la decisión a tomar
- $W_e$ : Matriz tecnológica con incertidumbre para las variables que corresponden al futuro

Los problemas estocásticos son especialmente complicados debido a su tamaño, que crece con el número de escenarios. Pero además la propia generación de los escenarios y la definición de probabilidades a los mismos $p_e$ es complicado.

Una de las principales ventajas de la programación estocástica es que las soluciones que obtiene para los periodos congelados (x) son estables ante los diferentes escenarios.

Existen paquetes informáticos específicamente dedicados a la resolución de problemas con escenarios, pero también se pueden utilizar paquetes estándares, aunque la complejidad anteriormente aludida es una limitación evidente en su uso.

# 9 PROCEDIMIENTOS DE RESOLUCIÓN DE MODELOS DE PROGRAMACIÓN MATEMÁTICA

## 9.1 Introducción

Hasta el momento se han planteado procedimientos de Modelado, dando por supuesto que el modelo en sí mismo podría ser resuelto. En el presente apartado se pretenden mostrar cómo se resuelven problemas de programación matemática.

Pero existen en el mercado, disponibles para su adquisición, procedimientos generales de resolución capaces de resolver en tiempos adecuados problemas cada vez más complicados.

Aunque en la práctica es posible que sea más rentable diseñar el propio procedimiento de resolución. Por ello se abordan en la fase final del capítulo métodos para diseñar procedimientos de resolución para los problemas que ya han sido modelados

## 9.2 Resolución de problemas de programación matemática mediante el uso de paquetes ya disponibles

### 9.2.1 Algoritmos y paquetes

"Algoritmo" es una palabra de origen árabe, concretamente, *Al-jwarizmi* era el apodo de un matemático cordobés de nombre Mohamed ibn Musa. Un algoritmo es un conjunto finito y ordenado de operaciones cuya aplicación permite resolver un problema.

Un algoritmo puede ser programado como un conjunto de rutinas de ordenador para resolver problemas que tienen que estar expresados en un tipo particular de formato. Cuando los algoritmos resuelven problemas suficientemente generales se suelen agrupar en paquetes que posteriormente se comercializan.

Estos paquetes comerciales contienen habitualmente algoritmos para resolver modelos de Programación Lineal y de Programación Entera Mixta.

Cuatro son los tipos de algoritmos que se pueden encontrar generalmente en un paquete comercial:

1. Versiones revisadas del Simplex
2. Extensiones del Simplex para modelos separables
3. Algoritmos de Ramificación y Corte para modelos de Programación Entera
4. Algoritmos del Punto Interior

En cualquier caso otros hacen un uso cruzado de los diferentes métodos. Así la estrategia que parece más adecuada para problemas de muy grandes dimensiones es utilizar algoritmos genéricos que rápidamente nos conduzcan al entorno del óptimo y finalizar la exploración del espacio de soluciones con un Simplex que buscar el óptimo global del problema.

Una de las mayores ventajas del uso de "paquetes" comerciales es que son muy flexibles en su uso. Además suelen contener unas características que los hacen muy interesantes, si se desean utilizar: Reducción, Soluciones iniciales, Cotas de Variables, Análisis de Sensibilidad.

**Reducción**

Algunos paquetes tienen procedimientos para detectar y eliminar redundancias en los modelos. De este modo se reduce el tamaño y por tanto el tiempo de resolución.

**Soluciones iniciales**

La mayor parte de los paquetes permiten que el usuario aporte una solución inicial. Si está suficientemente cerca del óptimo el tiempo de resolución se verá sustancialmente reducido. Esta herramienta es especialmente útil cuando se quiere probar el efecto de pequeños cambios en los valores del modelo.

**Cotas de variables**

En algunas ocasiones el valor de las variables está acotado del siguiente modo:

$$x \leq U \quad \text{ó} \quad x \geq L \text{ donde } U \text{ y } L \text{ son constantes.}$$

Aunque en la formulación convencional de los problemas de Programación Lineal, esta restricción sería asociada a una fila de la matriz de restricciones, es más eficiente considerar *U* o *L* simplemente como una cota. Existe una versión modificada del Simplex que permite trabajar con estas variables acotadas.

**Restricciones de Cotas Generalizadas**

Son bastante habituales las restricciones del tipo:

$$x_1 + x_2 + .... + x_n \leq M$$

Este tipo de restricciones corresponden con una fila de unos en la matriz de restricciones y se suelen denominar Cotas Superiores Generalizadas (GUB por *Generalized Upper Bound* en inglès) del subconjunto. Si existen muchas restricciones de este tipo para conjuntos disjuntos es útil representarlos como GUB's, puesto que reducen considerablemente el coste computacional.

**Análisis de sensibilidad**

Cuando se halla la solución óptima, a menudo nos interesa investigar como afectarían cambios en los coeficientes de la función objetivo y las restricciones. Esta información, que se puede obtener de todos los paquetes comerciales es de una gran importancia, en el uso de los modelos.

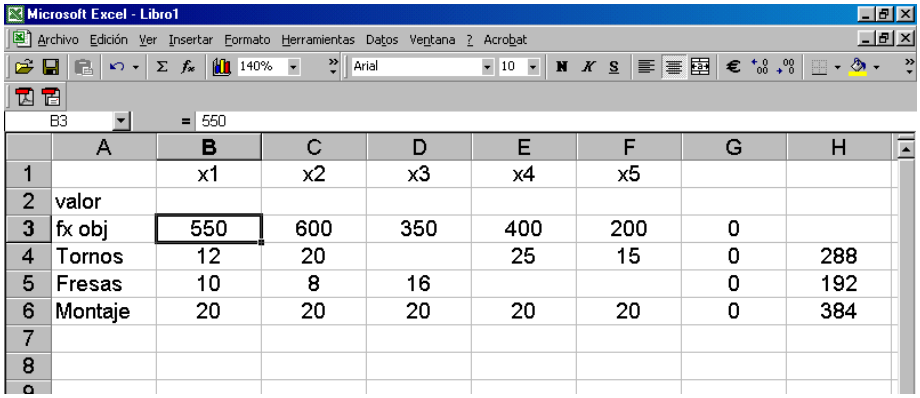
### 9.2.2 El uso de la Hoja de Cálculo Excel

La hoja de cálculo Excel dispone entre sus herramientas una dedicada a la resolución de problemas de Programación Matemática. Dicha herramienta (que en ocasiones hay que instalar a través de *Herramientas/Opciones*) se denomina *Solver*.

Para ilustrar su uso, bastante intuitivo por otro lado, se procede a plantear el modelo del apartado 3.6.

Una disposición de las celdas, meramente orientativa, pues el *solver* es versátil en ese aspecto, podría ser la siguiente:

En la celda G3 se ha introducido la fórmula "*=SUMAPRODUCTO($B2:$F2;B3:F3)"* ésta celda se ha copiado en las celdas de la columna G inferiores.

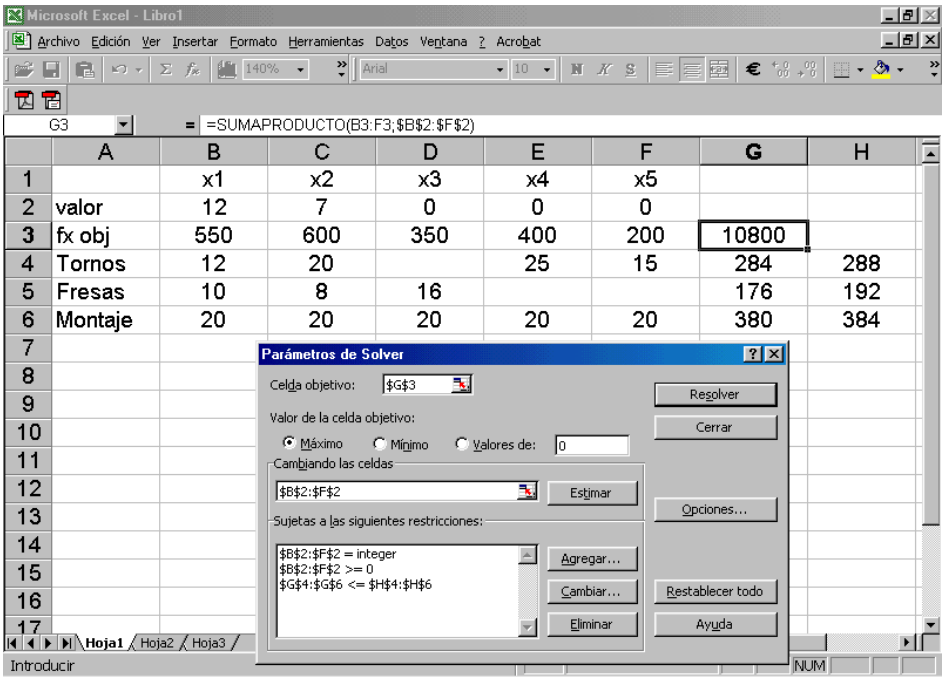



| | A | B | C | D | E | F | G | H |
|---|---|---|---|---|---|---|---|---|
| 1 | | x1 | x2 | x3 | x4 | x5 | | |
| 2 | valor | | | | | | | |
| 3 | fx obj | 550 | 600 | 350 | 400 | 200 | 0 | |
| 4 | Tornos | 12 | 20 | | 25 | 15 | 0 | 288 |
| 5 | Fresas | 10 | 8 | 16 | | | 0 | 192 |
| 6 | Montaje | 20 | 20 | 20 | 20 | 20 | 0 | 384 |
| 7 | | | | | | | | |
| 8 | | | | | | | | |

Figura X

Para poder usar el *solver* hay que acceder a través de *Herramientas/Solver*.

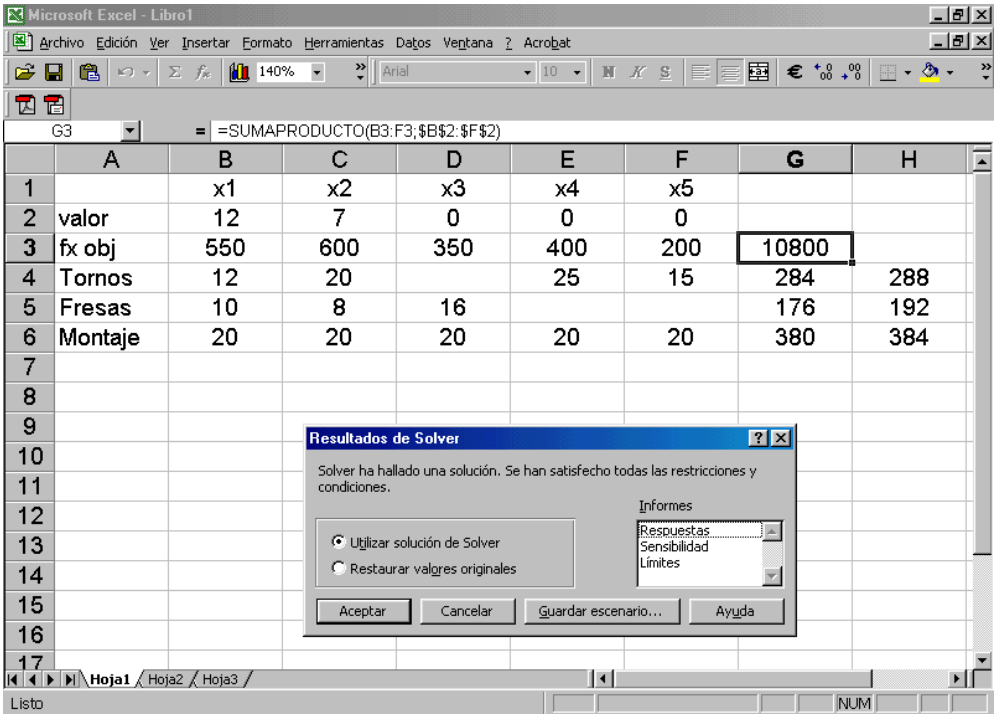
En la fila 1 se ha puesto los nombres de las variables y en la fila 2 aparecerán los valores de éstas. La fila 3 se dedica a los coeficientes de la función objetivo. Las filas 4 a 6 se dedican a las restricciones. En la columna H se colocan los límites de las restricciones.




| | A | B | C | D | E | F | G | H |
|---|---|---|---|---|---|---|---|---|
| 1 | | x1 | x2 | x3 | x4 | x5 | | |
| 2 | valor | 12 | 7 | 0 | 0 | 0 | | |
| 3 | fx obj | 550 | 600 | 350 | 400 | 200 | 10800 | |
| 4 | Tornos | 12 | 20 | | 25 | 15 | 284 | 288 |
| 5 | Fresas | 10 | 8 | 16 | | | 176 | 192 |
| 6 | Montaje | 20 | 20 | 20 | 20 | 20 | 380 | 384 |

Figura X

Para que el ejemplo sea más ilustrativo se ha incorporado una restricción de integridad. Accediendo al menú de opciones se pueden modificar parámetros tales como el método de resolución usado, o el tiempo máximo permitido.




| | A | B | C | D | E | F | G | H |
|---|---|---|---|---|---|---|---|---|
| 1 | | x1 | x2 | x3 | x4 | x5 | | |
| 2 | valor | 12 | 7 | 0 | 0 | 0 | | |
| 3 | fx obj | 550 | 600 | 350 | 400 | 200 | 10800 | |
| 4 | Tornos | 12 | 20 | | 25 | 15 | 284 | 288 |
| 5 | Fresas | 10 | 8 | 16 | | | 176 | 192 |
| 6 | Montaje | 20 | 20 | 20 | 20 | 20 | 380 | 384 |

Figura X

Al acabar la resolución del problema el *Solver* pregunta qué tipo de informes se desea. Y por tanto es posible acceder a ellos. Dichos informes incluyen respuestas, sensibilidad (precios-sombra y costes-reducidos) y análisis de límites (rangos)

### 9.2.3 El uso de Lenguajes de Modelado

Se pueden encontrar en el mercado diferentes paquetes que ayuden al usuario a estructurar e introducir un problema en un paquete en forma de modelo. Este tipo de programas se denominan "generadores de matrices" por su origen, aunque en realidad son lenguajes de alto nivel para la programación matemática.

El lenguaje de escritura más utilizado es MathProg que es implementado en diferentes herramientas como MPL. Algunas son de pago (AMPL y MPL de Maximal *Software* por ejemplo) y otras son gratuitas como . Gusek o LPSolveIDE.

## 9.3 El uso de paquetes para la resolución de modelos de Programación Lineal

Aunque, en principio, uno podría programarse sus propios algoritmos para la resolución de problemas de Programación Lineal, la realidad es que existen muchas alternativas que son mucho más eficientes que cualquier implementación individual.

En el terreno del *software* comercial dos son los paquetes relevantes: CPLEX y Gurobi. CPLEX es el *software* clásico. Llevan años mejorando sus algoritmos y los últimos pasos que dio (concretamente CPLEX 12) parecían haber dejado completamente atrás a sus competidores. Sin embargo Gurobi en el 2009 saltó a la arena, con un solucionador que es capaz de dejar atrás a CPLEX. En cualquier caso pueden resolver problemas con decenas de miles de variables, tanto enteras como binarias, en algunos segundos.

Muy lejos de estos dos *softwares* se encuentran las aplicaciones de *software* libre como GLPK LPSolve.

En cualquier caso además del *software* de optimización, generalmente hace falta tener un *software* de interface, para poder introducir datos en formato natural, y extraerlos de manera natural también.

Por otro lado, el uso de *software* para la resolución de programación lineal cuando los programas son de un cierto tamaño (unos miles de variables), puede requerir para ser eficiente un cierto nivel de "tuneado". Parámetros como los gaps admisibles, las reglas de pivote, la reducción previa, exigen unos minutos hasta dar con una combinación adecuada.

Se admite habitualmente que el mayor problema para aplicar de modo exitoso los modelos de programación matemática no son los modelos ni los tiempos de resolución, sino la interface con el usuario: la entrada y la salida de datos.

El uso de paquetes de modelado aporta las siguientes ventajas:

a) Ofrecer un modo de entrada de datos más natural

b) Corregir y verificar es mucho más fácil

c) Modificar se hace más fácilmente

d) Automatizar la repetición de la ejecución es factible

#### 9.3.1.1 Brevísima Introducción al MPL

El MPL es uno de los programas existentes en el mercado que facilita la introducción de modelos matemáticos en sistemas de resolución de problemas de programación matemática.

Una versión MPL de prueba para estudiantes puede descargarse de la web www.maximal-usa.com. Existiendo también en dicha web un muy buen manual en castellano, y múltiples ejemplos.

La utilidad del MPL se aprecia rápidamente sin más que analizar cómo se introduciría el problema del apartado 3.6 citado anteriormente.

***Ejemplo 16***

```
INDEX
  i= 1..5;
  j= (torno,fresa,montaje);
DATA
  benef[i] := (550,600,350,400,200);
  prodhoras[j,i] :=(12,20,0,25,15,
                    10,8,6,0,0,
                    20,20,20,20,20);
  rechoras[j] := (288,192,384);
VARIABLES
  x[i];

MAX SUM( i : benef*x);

SUBJECT TO
 restr[j] :  SUM(i : prodhoras*x)<=rechoras;

END
```

Es evidente la relación entre este modo de introducir el modelo y el propio modelo.

Además MPL incluye muchas otras funcionalidades como que Los datos pueden imputarse directamente, o a través de ficheros de texto, ficheros xml, Excel®, Access® y otros medios. O que Si hay sumatorios largos se pueden calcular por separado como macros.

Además de un fichero solución convencional se puede conectar a Excel® o Access®. Y por tanto el sistema puede dar y recibir información a/de paquetes convencionales.

Por otro lado el MPL tiene funcionalidades como al corrección ortográfica y sintáctica de los modelos, así como herramientas que permiten analizar los modelos desde un punto de vista estructural.

#### 9.3.1.2 El proyecto NEOS

También es relevante citar del denominado proyecto NEOS. Este se traduce en una web donde los mejores optimizadores están disponibles para ser utilizados remotamente. Además el servidor tiene interesantes resúmenes sobre la programación matemática. http://www-neos.mcs.anl.gov/neos/ es la dirección web donde toda esta mina de información está disponible.

### 9.3.2 Sistemas de Apoyo en la Decisión y Sistemas Expertos

En ocasiones, al diseñar un *software* para aplicaciones específicas, se incorporan algoritmos de programación matemática que sirven de soporte a la toma de decisión. Generalmente este *software* desarrollará otras múltiples funciones además de resolver el modelo, como acceder a bases de datos e

interactuar con el gestor o persona que toma la decisión. Si tales sistemas incorporan algoritmos que resuelvan los problemas más habituales y establecidos, el gestor podrá concentrarse en la toma de decisión real. Para diseñar dichos algoritmos es conveniente construir modelos y diseñar métodos de resolución de Programación Matemática.

A medida que los sistemas de apoyo a la toma de decisión se vuelven más sofisticados incorporan procedimientos que plantean "opciones" y no sólo representan datos. La definición de tales "opciones" no es más que encontrar soluciones a los problemas que se plantean y modelan.

Otro concepto relacionado es el denominado "sistema experto". Estos sistemas aceptan, en teoría, definiciones informales de los problemas, que se formalizan a medida que el usuario le ayuda, a través del interface diseñado al efecto, seleccionando entre diferentes alternativas. Esta información que añade el usuario se complementa con información histórica recogida de resoluciones anteriores.

## *9.4 Procedimientos de Resolución de Programación Lineal*

Se proponen en este apartado los principios en los que se basan el método de resolución conocido como Simples. Este método se utiliza desde su desarrollo a mediados de los 50. En la década de los 80 se desarrollaron nuevos métodos como los de Punto Interior o el Método de los Elipsoides, que no se tratarán en estos apuntes.

### 9.4.1 El algoritmo Simplex

No es objeto de este apartado establecer concretamente cómo funciona el algoritmo del Simplex. Sin embargo sí es necesario destacar los principios básicos de funcionamiento.

Dado un problema de Programación Lineal, éste se debe expresar de forma estándar (maximización y desigualdades menor-igual o minimización y desigualdades mayor-igual)

$$\begin{aligned} [Maximize] \quad & z = \sum c_i \cdot x_i \\ Sujeto\, a: & \\ & \sum_j a_{ij} \cdot x_j \le b_i \quad \forall i \end{aligned}$$

Las restricciones se pueden expresar mediante notación matricial de modo:

$$A \cdot x \le b$$

Se obtiene una solución básica al problema $A \cdot x \le b$ haciendo n-m variables iguales a 0 (para convertir el problema en linealmente independiente y determinado). A las variables definidas como nulas se les denomina variables no-básicas, y a las demás, variables básicas. Si existe solución para el problema planteado con las n-m variables no básicas definidas, entonces a esa solución se le denomina solución básica factible.

Las soluciones básicas factibles generan los vértices de un poliedro convexo. En uno de estos vértices se encuentra el óptimo si este existe. Se dice que dos vértices son adyacentes si sus conjuntos de variables básicas tienen m-1 variables básicas en común.

El algoritmo simplex consiste, básicamente, en desplazarse de un vértice a otro adyacente (empezando por uno cualquiera) hasta que cualquier movimiento "empeore" la función objetivo.

El algoritmo simplex clásico, ha sido revisado, en diferentes ocasiones. De él han salido procedimientos, denominados simplex revisados, en los que se hace un especial hincapié en la estructura matricial y por tanto en el cálculo matricial, reduciendo notablemente el coste computacional.

### 9.4.2 Los Métodos del Punto Interior

Los métodos del punto interior se basan en que el progreso de las soluciones se realiza en el interior de la región factible, y no a través de los vértices de la misma como hace el método Simplex.

La estrategia del algoritmo del punto interior es implementar, en el espacio original de la región de factibilidad, transformaciones a media que avanza el algoritmo.

Una vez conocida la región de factibilidad y un punto inicial e interior al mismo, se debe definir un modo de progreso en una dirección que permita reducir el valor de la función objetivo, y que el nuevo punto siga siendo factible e interior.

Este proceso se debe repetir hasta que no pueda haber una mayor reducción en el valor de la función objetivo.

Así pues las preguntas que hay que saber cómo responder son tres:

1) ¿Cómo encontrar un punto factible e interior?
2) ¿Cómo definir el camino de evolución?
3) ¿Cómo se sabe que se ha alcanzado el óptimo?

## *9.5 Procedimientos de Resolución en Programación Lineal Entera*

### 9.5.1 Ramificación y acotación

En la práctica la mayor parte de los problemas de PE se resuelven utilizando la técnica conocida como ramificación y corte (Ramificación y Acotación, Ramificación y Poda ó *Branch and Bound en inglés*). Este procedimiento se basa en la división de problemas en sub-problemas (ramificación) mientras el óptimo no sea una solución factible entera.

La Ramificación y Poda es un método general de búsqueda. Empieza considerando el problema raíz (el problema original con la región factible completa sin las restricciones de integridad). Si se cumplen dichas restricciones el procedimiento ha terminado. Si no se cumplen, la región de factibilidad se divide en o más regiones. Se resuelve cada uno de los sub-problemas, 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, su valor se puede tomar como una cota. Si el valor de dicha cota es peor que el mejor resultado obtenido hasta el momento, no hará falta seguir explorando este problema. Este procedimiento se resuelve de modo iterativo para cada uno de los sub-problemas generados.

Concretamente en Programación Entera, el método consiste en:

- Relajar el problema de PE convirtiéndolo en Programación Lineal,
- Resolver mediante el simplex este problema de Programación Lineal,
  - Si las variables de la solución óptima satisfacen la condición de integralidad, hemos obtenido el óptimo.
  - Si dicha condición no se satisface, el valor de la función objetivo aporta una cota superior del óptimo. Entonces, hay que volver a ramificar y volverlo a intentar.

***Ejemplo 17***

$$
\begin{aligned}
&[Maximize] \quad z = 7x_1 + 3x_2 + 4x_3 \\
&\quad Sujeto\ a: \\
&\qquad 3x_1 + 2x_2 - x_3 \le 5 \\
&\qquad\quad 2x_2 - 3x_3 \le 9 \\
&\qquad 7x_1 + 2x_2 \qquad\ \le 8 \\
&\qquad x_1, x_3 \in \mathbb{Z}^+ \\
&\qquad x_2 \ge 0
\end{aligned}
$$

La solución del problema relajado es

$$
\begin{aligned}
&(x_1, x_2, x_3) = (1.1428, 0, 3) \\
&z = 14.2857
\end{aligned}
$$

Este valor es una cota máxima del problema, aunque no es un valor solución porque $x_1 \notin \mathbb{Z}^+$

La ramificación consiste en generar un par de nuevos sub-problemas, mediante la adición respectiva de nuevas restricciones.

Sub-problema2 = Problema1 + $(x_1 \le 1)$

Sub-problema3 = Problema1 + $(x_1 \ge 2)$

La relajación del sub-problema2 da lugar a la siguiente solución y función objetivo

$$(x_1, x_2, x_3) = (1, 1.5, 2.667) \qquad z_2 = 14.1667$$

Dado que $x_2 \notin \mathbb{Z}^+$, volvemos a obtener una cota máxima más ajustada aunque la solución no es factible.

La relajación del sub-problema3 no tiene solución, por lo que no se profundizará más en esa rama.

Se crearán pues dos nuevos sub-problemas

Sub-problema4 = Sub-problema2 + $(x_3 \leq 2)$

Sub-problema5 = Sub-problema2 + $(x_3 \geq 3)$

La relajación del sub-problema 4 da la siguiente solución.

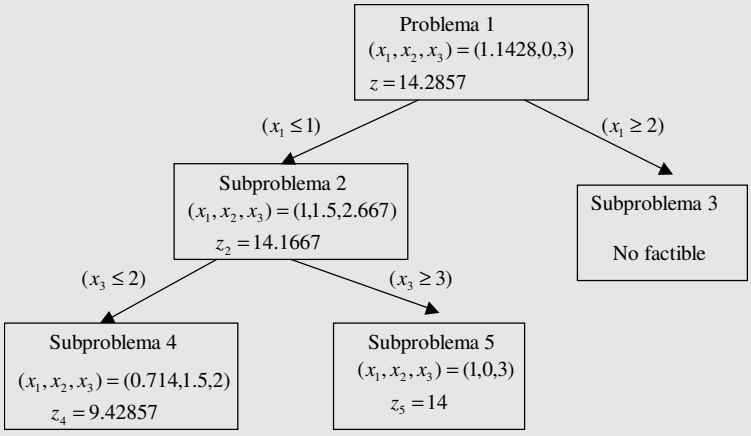
$$(x_1, x_2, x_3) = (0.714, 1.5, 2) \qquad z_4 = 9.42857$$

La relajación del sub-problema 5 da la siguiente solución.

$$(x_1, x_2, x_3) = (1, 0, 3) \qquad z_5 = 14$$

La solución del sub-problema 5 da valores enteros de $x_1$, $x_3$, y además se puede garantizar su optimalidad porque la cota superior del sub-problema 4 es 9.42, inferior al 14 ya obtenido.

Gráficamente se puede representar del siguiente modo:




En la resolución mediante métodos de ramificación y acotación, se puede optar básicamente por dos estrategias en la selección del siguiente nodo a ramificar:

a) Elegir la rama más profunda
b) Elegir la rama con mejor cota
c) Elegir el primero de los nodos aún no explotados (FIFO)
d) Elegir el último nodo abierto y no explotado (LIFO)

Aunque aparentemente la opción 'b' llevaría más rápidamente a la solución, lo cierto es que depende de cada problema la selección de una alternativa u otra.

Otra técnica interesante en la aplicación de la ramificación y corte es la del cálculo inicial de una solución que permita fijar la cota superior (en algunos textos anglófonos se denomina *incumbent*). Ésta, obtenida mediante algún procedimiento heurístico, daría una solución candidato que permitiría, en ocasiones, podar ramas sin necesidad de explorarlas profundamente.

Por otro lado, aunque en este apartado se ha explicado la técnica de ramificación y acotación para un problema de PE, es posible aplicarlo en otros entornos conservando la misma configuración (Por ejemplo para la resolución de problemas de Optimización Combinatoria).

### 9.5.2 Enumeración implícita

El método de enumeración implícita se usa frecuentemente para resolver PE 0-1. En la enumeración implícita se aplica el hecho de que cada variable es binaria para simplificar las componentes a ramificar y acotar del proceso de ramificación y acotación y para determinar eficazmente cuando un nodo no debe ser explorado.

El árbol que se usa en el método de la enumeración implícita es similar a los árboles que se utilizan para resolver PE convencional. Cada rama del árbol especificará para una determinada variables si vale $x_i=0$ ó $x_i=1$. Todas las variables a las que se las asigna un valor se denominan variables fijas y las demás variables libres.

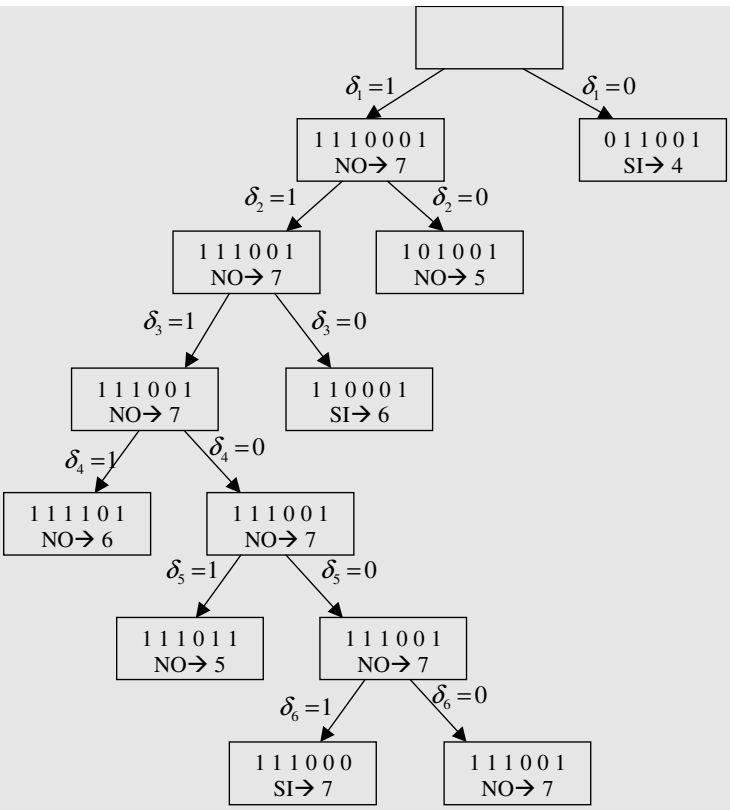
Para cualquier nodo (un conjunto de variables fijas con sus valores asignados) se denomina integración del nodo a cualquier especificación de las variables libres.

Un modo de integrar de modo eficiente un nodo es asignando a las variables libres los valores que permiten maximizar (o minimizar) la función objetivo. Si esta integración es factible entonces esta es la mejor solución a que se puede llegar a través del nodo en el que nos encontramos. Si no fuera factible el valor de la función objetivo obtenido definiría una cota superior (o inferior) del nodo.

Desgraciadamente no hay ningún modo efectivo de saber si para un nodo no existe ninguna integración factible.

***Ejemplo 18***

$$\begin{aligned} &[Maximize] \quad 3\delta_1 + \delta_2 + \delta_3 - \delta_4 - 2\delta_5 + \delta_6 \\ &\quad Sujeto\,a: \\ &\quad\quad \delta_1 + \delta_2 - 2\delta_3 + \delta_4 \leq 2 \\ &\quad\quad \delta_1 + \delta_2 + \delta_3 + \delta_5 \leq 3 \\ &\quad\quad \delta_1 + \delta_3 + \delta_5 + \delta_6 \leq 2 \end{aligned}$$




### 9.5.3 Métodos del plano cortante

Los métodos del plano cortante se utilizan para resolver problemas de PE. El método consiste en relajar los problemas de PE a problemas de Programación Lineal. Si la solución resultante del problema es entera, el problema ha sido resuelto.

Si no es así, se planteará una nueva restricción (plano de corte) que se añadirá al problema y que pretenden "relajar" las aristas que dan soluciones no enteras.

El nuevo problema se vuelve a resolver como Programación Lineal y el proceso anterior se repite hasta que la solución es entera.

Aunque el número de cortes necesarios para garantizar la optimalidad se ha demostrado que es finito, en realidad no es un procedimiento muy exitoso para problemas grandes.

## *9.6 Procedimientos de Resolución en Programación 0-1*

Cuando el número de variables binarias es muy alto, probablemente compense diseñar procedimientos específicos como los explicados en el capítulo de Optimización Combinatoria que sucede a éste.

# 10 OPTIMIZACIÓN COMBINATORIA. INTRODUCCIÓN

## 10.1 Introducción

Como se ha comentado anteriormente el Ingeniero de Organización Industrial diseña y mejora artefactos lógicos. Estos suelen tener forma de herramienta *software* que interpreta datos en el ámbito de un problema, y los transforma entregando como resultados datos que pueden ser ejecutados.

Específicamente dentro del área de la Gestión de la Producción y las Operaciones, muchos de los problemas que se plantean tienen estructuras especialmente difíciles para garantizar la optimalidad en la resolución.

Entre las herramientas disponibles para generar dichas soluciones, los algoritmos de optimización combinatoria resuelven instancias de problemas que se creen difíciles en general, explorando el espacio de soluciones (usualmente grande) para estas instancias. Los algoritmos de optimización combinatoria logran esto reduciendo el tamaño efectivo del espacio, y explorando el espacio de búsqueda eficientemente.

La mayor parte de los procesos de resolución de problemas en optimización combinatoria no garantizan el óptimo ni siquiera en el contexto de modelo, que no necesariamente es el problema real, pero su aproximación al óptimo es probablemente suficiente.

Los métodos de resolución se pueden clasificar de muchas maneras. Una de ellas permite pueden clasificar los métodos de resolución en 4 grandes grupos:

a) Algoritmos constructivos: Construyen la solución desde una solución incompleta, a partir de los datos del problema y el conocimiento que del mismo se tiene.

b) Algoritmos de mejora: Comienzan con una solución factible y completa al problema, y esta solución general es modificada gradualmente de un modo progresivo.

c) Estrategias de "divide y vencerás": En esta estrategia se opta primero por dividir el problema en fragmentos a los que se le aplica cualquier otra estrategia, recomponiendo finalmente la solución.

d) Estrategias de aprendizaje: Las estrategias de aprendizaje pasan por tomar decisiones, en función de datos conocidos por el resultado de resoluciones anteriores o en la misma resolución.

La clasificación que se utiliza aquí extiende el concepto de heurística tanto para los métodos constructivos como para los de mejora, y considera las últimas dos estrategias las dentro del proceso general de resolución, más que entre los procedimientos de resolución en Optimización Combinatoria.

Los métodos habitualmente utilizados para resolver problemas son del tipo heurístico o meta-heurístico. Estos son capaces de generar soluciones al problema. Son aproximaciones que pretenden acercarse lo más posible al óptimo, pero que pueden fallar en el intento. En los inicios de la Investigación Operativa, y dada las limitaciones de cálculo automático que entonces se tenía, era

habitual desarrollar procedimientos heurísticos (es decir que hallaban soluciones) muy rápidamente. Los procedimientos eran específicos para cada problema y en general su eficiencia era escasa cuando se intentaban utilizar en otros ámbitos.

Pese a no garantizar la optimalidad, los métodos heurísticos de resolución (incluyendo entre ellos los procesos de mejora local y los algoritmos meta-heurísticos) son básicos por varios motivos. En primer lugar son capaces de generar soluciones lo que generalmente es mejor que no tener solución alguna. En segundo lugar se puede decir que alcanzar el óptimo de un modelo que tampoco es exactamente el problema real no es esencialmente grave. Por último ser capaz de diseñar una buena heurística, exige un conocimiento del problema que puede conducir a mejoras de otro tipo.

Con la existencia de ordenadores cada vez más potentes, y aprovechando su capacidad de cálculo han surgido en las últimas décadas, las denominadas técnicas meta-heurísticas. Son procedimientos genéricos que pueden ser fácilmente adaptados para resolver problemas de optimización combinatoria.

La mayoría de las meta-heurísticas tienen como objetivo los problemas de optimización combinatoria, pero por supuesto, se pueden aplicar a cualquier problema que se pueda reformular en términos heurísticos, por ejemplo en resolución de ecuaciones booleanas.

Muchos de los procedimientos denominados meta-heurísticos se fundamentan generalmente en la capacidad del mundo natural de encontrar buenas soluciones y se intenta reproducir esta capacidad, surgiendo de este modo algoritmos como el Recocido Simulado, los Algoritmos Genéticos, las Redes Neuronales, los Sistemas de Hormigas y *GRASP* (Diaz, Glover, Ghaziri, Gonzales, Laguna, and Moscato, 1996).

En realidad los fenómenos naturales en los que dicen basarse todas las meta-heurísticas tienen en común su capacidad de explorar convenientemente el adyacente posible (Kauffman, 2003) para abandonar óptimos locales.

Se puede realizar una clasificación simplificada de los procedimientos meta-heurísticos indicando que hay tres grandes grupos. El primer grupo lo forman los que dada una solución intentan mejorarla (búsqueda local). El segundo grupo lo conforman los que parten de un conjunto de soluciones (población) para intentar obtener una mejor por combinación y evolución de las anteriores. El tercer grupo lo conforman aquellos que pretenden específicamente aprender de soluciones previas.

La búsqueda local es la base de la mayor parte de los métodos heurísticos. Empezando por una determinada solución (en ocasiones generada aleatoriamente) la búsqueda local se mueve hacia un vecino que es mejor que la actual solución de acuerdo con la función objetivo. En general la búsqueda local corre el riesgo evidente de quedar atrapada en un óptimo local y la mayor parte de los algoritmos basados en óptimos locales pretenden evitar este problema.

Estos apuntes se estructuran en dos partes, una dedicada a la introducción a la Optimización Combinatoria y otra a los procedimientos de resolución. Se acompañan los apuntes con resoluciones aproximadas a 3 problemas según serían requeridos en examen.

En la primera parte se pretende, en primer lugar, establecer qué es la Optimización Combinatoria y dar una idea general sobre la complejidad de establecer soluciones óptimas en dichos entornos. Se presenta a continuación una aproximación a los problemas de optimización combinatoria mediante una tentativa de clasificación de los mismos. Se presenta finalmente una aproximación a los criterios que permiten seleccionar algoritmos en función de los requerimientos solicitados.

En la segunda parte se abordan los procedimientos de resolución de problemas de optimización combinatoria clasificados en 3 grandes grupos:

a) Procedimientos de Resolución Aleatorios

b) Procedimientos Enumerativos

c) Procedimientos Heurísticos (se incluyen en estos los Meta-heurísticos

## *10.2 Complejidad en la Optimización Combinatoria*

### 10.2.1 Optimización Combinatoria

Un problema de optimización combinatoria es un problema de optimización en el cual el espacio de soluciones posibles es discreto.

La optimización combinatoria es una rama de la optimización de las matemáticas aplicadas fuertemente relacionada con la investigación operativa, la teoría algorítmica y la teoría de la complejidad computacional. Los algoritmos de optimización combinatoria resuelven problemas que se creen difíciles en general, por la vía de explorar el, habitualmente grande, espacio de soluciones del citado problema. Los buenos algoritmos de optimización combinatoria lo logran por la vía de reducir el tamaño efectivo del espacio a buscar y explorando de modo eficiente dicho espacio.

### 10.2.2 Variaciones (con y sin) Repetición, Combinaciones, Permutaciones

Es relevante recordar algunos conceptos de combinatoria, pues estos permiten establecer la dimensión de alguno de los problemas aquí tratados.

Los problemas de combinatoria se pueden definir como la “Selección de un sub-conjunto de entre un conjunto de elementos con permitiendo (o no) la repetición y siendo (o no) relevante el orden de los mismo”.

Si el subconjunto seleccionado es todo el conjunto original el problema tratado es de permutaciones. Si el orden no importa el problema es de Combinación.

A continuación se recuerdan algunas fórmulas básicas que permiten calcular el número de alternativas posibles a determinadas estructuras.

**Variaciones sin Repetición:** Número de modos de seleccionar y ordenar *r* elementos de un conjunto de *n*, donde ninguno se puede repetir. $A_n^r = \frac{n!}{(n-r)!}$

**Variaciones con Repetición:** Número de modos de seleccionar y ordenar *r* elementos de un conjunto de *n*. $n^r$

**Permutación sin Repetición:** Número de modos en el que se puede ordenar un conjunto de *n* elementos distintos entre sí. $n!$

**Permutación con Repetición:** Número de modos de ordenar un conjunto de *n* elementos donde hay *m1* elementos de tipo *1, m2* elementos de tipo *2…mS* elementos de tipo S. $\frac{n!}{m1!\cdot m2!\cdot\cdots\cdot mS!}$

**Combinaciones sin Repetición:** Número de modos de seleccionar *r* elementos de un conjunto de *n*, donde ninguno se puede repetir. $C_n^r = \frac{n!}{r!\cdot(n-r)!}$

**Combinaciones con Repetición:** Número de modos de seleccionar *r* elementos de un conjunto de *n*, permitiendo la repetición en la selección. $C_{n+r-1}^r = \frac{(n+r-1)!}{r!\cdot(n-1)!}$

Por su interés se destacan algunas otras relaciones que permiten calcular de cuantos modos se pueden introducir elementos (iguales o distintos-ordenados y no ordenados) en casillas (iguales o distintas – ordenadas y no ordenadas). Cuando se indica que los elementos están ordenados hace referencia a que interesa el orden, no a que estén previamente ordenados.

***r* Objetos distintos y no ordenados en *n* casillas distintas y no ordenadas** $n^r$

***r* Objetos idénticos en *n* casillas distintas y ordenadas** $C_{n+r-1}^r = \frac{(n+r-1)!}{r!\cdot(n-1)!}$

***r* Objetos distintos y ordenados en *n* casillas distintas y ordenadas** $C_{n+r-1}^r = \frac{(n+r-1)!}{(n-1)!}$

Todas estas relaciones (junto con otras muchas y –lo que es más importante- el modo de llegar a ellas), se pueden encontrar en (Kauffmann, 1971).

### 10.2.3 Las clases de complejidad P y NP

**Teoría de la Complejidad**

La teoría de la complejidad computacional es la rama de la teoría de computación que estudia los recursos, o coste, de computación requerido para resolver un problema dado.

Este coste se suele medir en términos de parámetros abstractos como el tiempo y el espacio. Tiempo son los pasos que requiere el problema para ser resuelto y el espacio suele referirse a la cantidad de memoria que utiliza.

Uno de los otros recursos que se puede considerar es el número de procesadores en paralelo que se necesitan para resolver un problema en paralelo. En este caso hay que considerar el tiempo "paralizable" y el "no-paralelizable".

La teoría de la complejidad no es la teoría de la "computabilidad" que tiene que ver con la capacidad de resolver un problema, independientemente de los recursos requeridos.

**P y NP**

La clase de complejidad P es el conjunto de problemas de decisión que pueden ser resueltos por una máquina determinista en tiempo polinomial. Esta clase corresponden a la idea intuitiva de los problemas que se puede resolver de modo efectivo en los peores casos.

La clase de complejidad NP es el conjunto de problemas de decisión que pueden resolverse por una máquina no-determinista en tiempo polinomial. Esta clase contiene una gran cantidad de problemas que se pretenden resolver. La principal propiedad de los problemas de esta clase es que sus soluciones pueden ser chequeadas de modo eficiente en tiempo polinomial.

## *10.3 Algoritmos básicos.*

- Ordenar
- Buscar el Mayor (o menor)
- Intercambiar
- Sumar
- Voltear

## *10.4 Problemas de Optimización Combinatoria*

En este apartado se describen algunos problemas clásicos de optimización combinatoria. La intención inicial al redactar el apartado era establecer una clasificación, pero esto parece casi imposible, habida cuenta de que se pueden encontrar muchos tipos de clasificaciones en función de diferentes criterios para cada tipo de problema. Así, se ha optado por enumerar (sin ánimo de ser exhaustivos ni exclusivos) diferentes tipos de problemas en función de diferentes aspectos como por ejemplo: su aplicación, la clase teórica a la que pertenecen o las soluciones que se pretenden conseguir. Se han dejado aparte clasificaciones que considerarían aspectos como su dureza en términos de complejidad, o la existencia de algoritmos eficientes para su resolución.

### 10.4.1 Según su aplicación.

10.4.1.1 Secuenciación

Los problemas de secuenciación son, junto con los de rutas, la aplicación más clásica en optimización combinatoria, tanto por la extensión de su aplicación, como por su facilidad de planteamiento. Aun así el número de problemas diferentes que se puede plantear son muchos.

Definir el orden en el que se deben ejecutar las tareas de un proyecto para minimizar su duración, teniendo en cuenta limitaciones en los recursos, es un clásico problema de secuenciación.

La secuenciación en una máquina hace referencia al orden en que un conjunto de trabajos deben pasar por una máquina. La secuenciación puede considerar aspectos como la duración, fecha de entrega, la importancia del cliente…

Los problemas de secuenciación de una máquina se extienden teniendo en cuenta máquinas en paralelo y/o máquinas en serie (taller de flujo) y de ahí al taller general. Otra extensión habitual es considerar tiempos de preparación de máquinas. Además el uso de múltiples recursos diferentes y limitados forma parte también de las posibilidades.

Si además de definir la secuencia (o incluso asignar la máquina) se tuviera que definir la cantidad a producir se entraría en los problemas que unen la lotificación con la Secuenciación (ELSSP).

La secuenciación con mezcla de modelos en una línea de montaje, pretende establecer el orden en el que un conjunto de productos (pertenecientes a varias familias de productos en diferentes cantidades) deben circular por una línea de montaje.

#### 10.4.1.2 Rutas

Los problemas de rutas tratan de establecer el circuito a recorrer para dar un determinado servicio, ya sea de entrega de recepción o de ambos. El problema clásico denominado del viajante de comercio (TSP) supone visitar una y sólo una vez un conjunto de puntos. A este problema se le pueden añadir variantes como incluir varios viajantes, limitar la capacidad del variante o la disponibilidad de tiempo, incorporar ventanas temporales de entrega o recepción, obligar o prohibir determinados tramos, hacerlo en una o varias ocasiones por cada tramo…

#### 10.4.1.3 Corte y Empaquetado

Los problemas de corte son los ligados a reducir el consumo de materia prima que se vende o consume troceada. Pueden ser unidimensionales (vigas), bidimensionales (placas de vidrio, cartón o tela) o incluso tridimensionales como en el corte de mármol. También se pueden considerar variantes como que las piezas a obtener tengan una forma regular, o irregular (por ejemplo retales de tela). O restricciones que obligan a mantener determinadas direcciones de corte, o realizar cortes completos.

Los problemas de *Trim* son una variante de estos problemas en los que de un rollo (de papel o de acero) se trocean otros rollos de anchos y longitudes diferentes). El problema radica en este caso en establecer los modos en los que los diferentes rollos se van haciendo para minimizar no sólo la materia prima desperdiciada sino también las preparaciones de máquina a realizar.

Los problemas de empaquetado son similares siendo los contrarios, pretenden ubicar en la mínima superficie (o el mínimo volumen) posible un conjunto de partes que son inicialmente diferentes.

#### 10.4.1.4 Horarios

Los problemas de horarios son un problema clásico de optimización de combinatoria que tiene en la gestión docente (aunque evidentemente no es exclusiva de ella) una gran cantidad de aplicaciones: el diseño de horarios de clase, repartos de guardias, calendarios de exámenes, son algunos fácilmente

reconocibles. En la industria se pueden encontrar fácilmente cuando se trata de repartir cargas de trabajo desiguales entre operarios con calendarios laborales.

10.4.1.5 Asignación

Los problemas en los que se asignan recursos a tareas o tareas a agrupaciones son otra clase habitual de problemas. El equilibrado de líneas es un problema muy conocido, pero los problemas de asignación de frecuencias o de generación de equipos multidisciplinares equilibrados no son menos habituales.

### 10.4.2 Según su clasificación formal

- El problema de la mochila (uni o multi-dimensional)
- De máximo (o mínimo) flujo
- Problemas de Asignación Cuadrática
- *Graph Partitioning Problem*
- *Bipartite Drawing Problem*
- Cubrimiento, Partición y Empaquetado.
- De camino más corto (o más largo)
- El problema de coloreado de grafos
- *Linear Ordering Problem*

### 10.4.3 Según las soluciones que se buscan

En algunos problemas, sólo se tiene interés en encontrar una solución factible (e incluso en algunos casos únicamente se pretende saber si existe una solución). En otros casos el objetivo es que la solución factible encontrada sea suficientemente buena o incluso que sea óptima (es decir que no haya ninguna solución mejor). Por último es posible que se tenga interés en localizar todas las soluciones óptimas o incluso todas las soluciones factibles.

## *10.5 Evaluación de Procedimientos*

Los algoritmos se pueden medir en función de su capacidad de alcanzar buenos resultados, y del coste (tanto en tiempo como en memoria que requieren) para alcanzar dichos resultados.

Estos criterios se denominan: Eficiencia, Robustez y Bondad.

Al comparar los algoritmos se puede realizar tanto en el tiempo de resolución que emplean como en el resultado que obtienen.

En el caso de comparar los resultados estos se pueden comparar contra el óptimo, contra una cota del óptimo, o contra el resultado de otros algoritmos. Lo que se mide puede ser el comportamiento promedio, el comportamiento en el peor de los casos o el número de éxitos (número de veces que se obtiene el óptimo o el mejor resultado).

# 11 OPTIMIZACIÓN COMBINATORIA. LA BÚSQUEDA RÁPIDA DE SOLUCIONES

## 11.1 Introducción

La mayor parte de los procesos de resolución de problemas en optimización combinatoria no garantiza el óptimo ni siquiera en el contexto de modelo, que no necesariamente es el problema real.

Foulds (1983) clasifica los métodos de resolución en 4 grandes grupos:

a) Algoritmos constructivos. Construyen la solución desde una solución incompleta, a partir de los datos del problema y el conocimiento que del mismo se tiene.

b) Algoritmos de mejora. Comienzan con una solución factible y completa al problema, y esta solución general es modificada gradualmente de un modo progresivo.

c) Estrategias de "divide y vencerás". Esta estrategia se opta primero por dividir el problema en fragmentos a los que se le aplica cualquier otra estrategia, recomponiendo finalmente la solución.

d) Estrategias de aprendizaje. Las estrategias de aprendizaje pasan por tomar decisiones mensuales robre el modo de resolver, en función de datos conocidos por el resultado de resoluciones anteriores o en la misma resolución.

Sin embargo la clasificación que se va a utilizar en estos apuntes extiende el concepto de heurística tanto para los constructivos como para los de mejora, y las últimas dos estrategias las considera dentro del proceso general de resolución, más que entre los procedimientos de resolución en Optimización Combinatoria.

Así se clasifican en estos apuntes los métodos según los siguientes cuatro tipos:

a) Generación de soluciones por métodos Heurísticos Constructivos

b) Generación de soluciones por métodos Enumerativos

c) Generación de soluciones por métodos Heurísticos de Mejora Local

d) Generación de soluciones por métodos Heurísticos de Población

Además se incorpora la generación de soluciones por métodos aleatorios, pues aunque son ineficientes en el proceso de búsqueda, pueden ser eficaces, tanto para calibrar métodos cómo, y más relevante para comprobar que se ha entendido bien el problema.

Los métodos habitualmente utilizados para resolver problemas son del tipo heurístico o meta-heurístico. Esto es, son capaces de generar soluciones al problema. Son aproximaciones e inacciones que pretenden acercarse lo más posible al óptimo, pero que pueden fallar en el intento.

Pese a su no-optimalidad, muchos de estos métodos son básicos por varios motivos. En primer lugar son capaces de generar soluciones lo que generalmente es mejor que no tener solución alguna. En segundo lugar se puede decir que alcanzar el óptimo de un modelo que tampoco es exactamente el

problema real no es esencialmente grave. Por último ser capaz de diseñar una buena heurística, exige un conocimiento del problema que puede conducir mejoras de otro tipo.

## *11.2 Procedimientos de Resolución Aleatorios*

### 11.2.1 ¿Para qué sirven?

Diseñar un procedimiento de resolución aleatorio debiera, en primer lugar, ser fácil de hacer. Además debiera permitir generar soluciones factibles (o al menos evaluables) de modo rápido. Dos son los motivos por los que se recomienda comenzar el análisis de cualquier problema con el desarrollo de procedimientos de resolución aleatorios:

a) Para comparar contra procedimientos más elaborados.

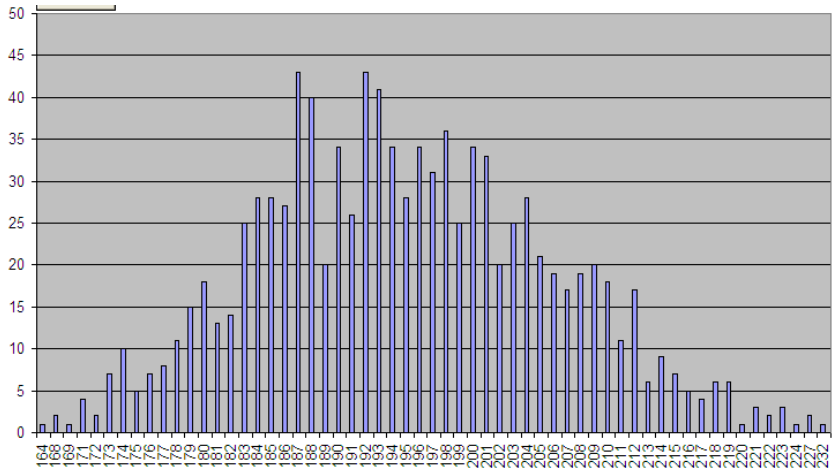
b) Para tener rápido un procedimiento que dé soluciones y comprobar que se está abordando el problema adecuado.

Además las soluciones obtenidas aleatoriamente pueden servir de base para otros procedimientos más elaborados (mejora local o algoritmos genéticos).

### 11.2.2 ¿Pueden funcionar?

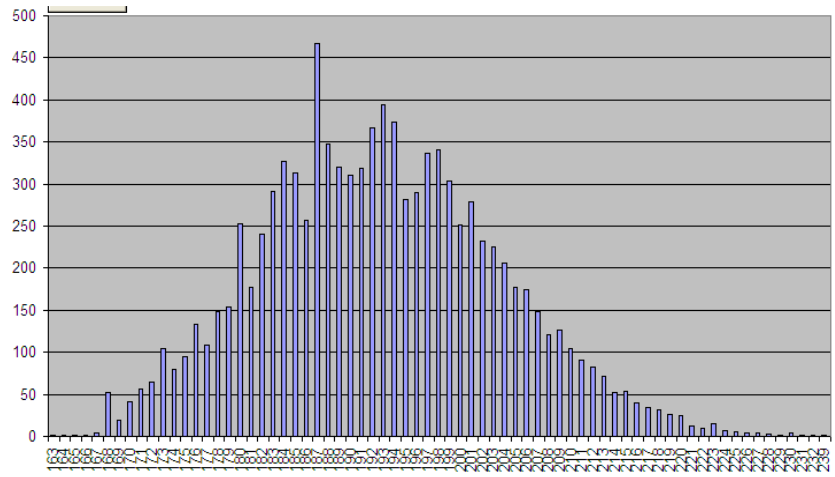
En principio esto dependería de cada problema, pero se puede generalizar diciendo que los resultados aleatorios tienen una baja probabilidad de ser buenos, pero en ese caso, la repetición suficiente podría llegar a dar, de un modo ni eficiente ni robusto, resultados aceptables.

Como ejemplo se ha evaluado un problema de *Flow-Shop* de 3 máquinas, 12 tareas, con tiempos aleatorios de ejecución entre 2 y 20. Se ha ejecutado un algoritmo aleatorio 1000 veces y se ha obtenido la siguiente gráfica de resultados.




Tras 1000 iteraciones en una ocasión se ha obtenido un resultado igual a 164, siendo el óptimo 163 (aunque eso no es conocido a priori).

Se ha repetido el experimento con 10000 iteraciones obteniendo los siguientes resultados.



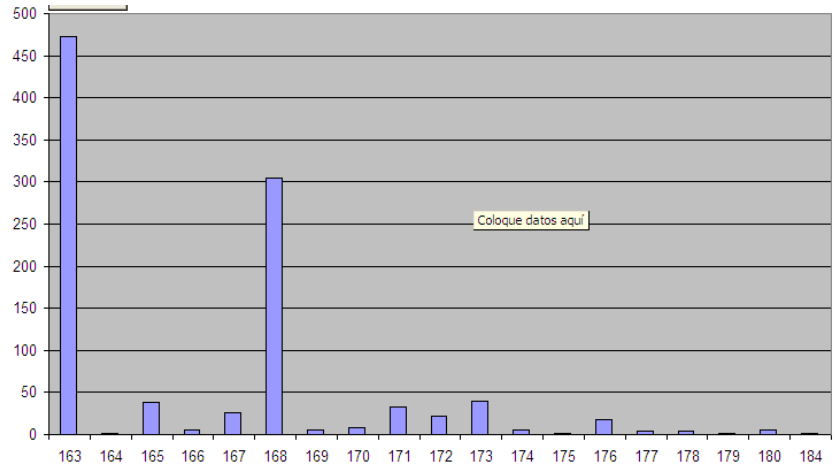
Es decir, en dos ocasiones (de entre 10000) se ha alcanzado el óptimo, y sólo en 10 ocasiones el error cometido sobre el óptimo ha sido menor que el 2,5%.

En problemas más grandes la dispersión crece, y además el tiempo de ejecución no es despreciable.

Sin embargo, y en el mismo problema, con un algoritmo de mejora iterativa sencillo, a partir de una solución aleatoria, se obtiene el óptimo en el 50% de las ocasiones, en la misma cantidad de tiempo que se tarda en generar aproximadamente 100 soluciones aleatorias.

Para comprobarlo se han utilizado las 1000 soluciones aleatorias generadas en la primera experiencia y a cada una de ellas se le ha aplicado un algoritmo de mejora iterativa con un vecindario definido según un intercambio de 2 (común para todos los problemas de permutación).

El histograma resultado ha sido el siguiente:




Es decir, el óptimo se alcanzó en casi el 50% de las ejecuciones y el peor resultado obtenido estaba a 13% del óptimo.

Con el anterior ejemplo se ha pretendido mostrar que las heurísticas aleatorias no funcionan de modo eficiente, aunque son fundamentales, pues permiten iniciar los procesos tanto de modelado como incluso de resolución.

### 11.2.3 Un procedimiento de generación aleatoria de soluciones

En este apartado se presenta un sencillo procedimiento que permite generar soluciones aleatorias para problemas en los que el vector solución es una permutación de n elementos.

Sea v un vector con n elementos a permutar

```
i:=0;
while i<n-1 do begin
    j:=random(1...n-i)
    intercambiar(v[i+1];v[i+j])
    i:=i+1
end;
```

Intercambiar es un procedimiento que intercambia los valores de v en las posiciones señaladas.

```
Procedure intercambiar(v[i+1];v[i+j])
begin
    aux:=v[i+1];
    v[i+1]:=v[i+j];
    v[i+j]:=aux;
end;
```

## *11.3 Algoritmos Heurísticos.*

### 11.3.1 Concepto de Heurística

Un algoritmo heurístico es un procedimiento que permite encontrar una solución y que suelen diseñarse de modo específico para cada problema. En métodos matemáticos, la palabra “heurística” suele hacer referencia a un procedimiento que busca una solución aunque no garantiza encontrar la mejor solución. En Inteligencia Artificial se suele denominar función heurística a aquella que dirige la búsqueda (o construcción) de una solución, utilizando algún mecanismo más o menos sencillo.

Un buen algoritmo heurístico debe ser eficiente, bueno y robusto. Esto es: debe requerir un esfuerzo computacional realista, su resultado debe estar suficientemente cerca del óptimo, y la probabilidad de obtener una mala solución debe ser baja.

Un tipo especial de heurísticas son las constructivas, es decir aquellas que en cada paso añaden un elemento más a una solución que no ha sido completamente construida. Las heurísticas constructivas se pueden utilizar siempre que el problema se puede resolver mediante decisiones sucesivas. (Rutas, Secuenciación, Líneas de Montaje…). En muchas ocasiones el objeto de la decisión es evidente, pero no necesariamente es así.

El otro tipo principal de heurísticas son las de búsqueda de entornos o de mejora local. Estas parten de una solución cualquiera (no necesariamente aleatoria) y avanzan buscando en el vecindario más próximo, produciendo mejoras hasta que alcanzan un punto donde ningún elemento en el vecindario es mejor que la solución de la que ya se dispone.

Las heurísticas también pueden ser de tipo voraz (en inglés *greedy*). Estas heurísticas eligen siempre las soluciones mejores de modo local para generar el resultado. Algunas heurísticas *greedy* son óptimas

(como ejemplo el algoritmo de Johnson para un *Flow-Shop* de 2 máquinas), pero no necesariamente es así. Los algoritmos de tipo voraz suelen ser muy rápidos pues no consideran alternativas.

La alternativa a las heurísticas *greedy* son las Heurísticas con *BackTracking*. Estas son heurísticas que si alcanzan un punto de no retorno (o no suficientemente buenos) retroceden en el proceso de reconstrucción, para analizar caminos abandonados.

- Métodos Constructivos
  - Voraces
  - Heurísticas con *BackTracking*
- Heurísticas No-Constructivas
  - Exploración de Entornos
- Métodos Combinados

### 11.3.2 Procedimientos Constructivos

Las Heurísticas constructivas se pueden utilizar siempre que el problema se puede resolver mediante decisiones sucesivas (Rutas, Secuenciación, Líneas de Montaje…). En muchas ocasiones el objeto de la decisión es evidente, pero no necesariamente es así.

Aunque cada heurística constructiva es diferente, y no hay ningún límite en su diseño, algunas de las alternativas que permiten plantear heurísticas son las siguientes:

a) **Reglas de Prioridad.** Se asignan valores a los objetos que se van a seleccionar en el proceso de construcción y se utilizan para elegir la siguiente opción. Los valores pueden calcularse de una vez para el resto del cálculo o variar dinámicamente tras cada decisión dando lugar a las Reglas Estáticas o a las Dinámicas. En muchos casos es importante que la regla de prioridad tenga en cuenta criterios de desempate.

b) **Dirección de Construcción.** En ocasiones es posible identificar una dirección de construcción. En general se suele trabajar hacia adelante, pero también es posible diseñar heurísticas que funcionen hacia atrás o incluso en las dos direcciones simultáneamente.

c) **Técnicas de Reducción Previa.** El análisis previo del problema puede llevar a tomar decisiones que afecten a la estructura de la solución antes de comenzar el proceso de resolución (Por ejemplo eliminar sub-secuencias antes de empezar por ser demasiado caras).

d) **Heurística de Paso Múltiple.** Se repite el cálculo cambiando las reglas de decisión o incluso tomando decisiones de modo estocástico.

e) **Heurísticas *MultiStart*.** Repetir la heurística pero comenzando cada vez por un elemento diferente.

#### 11.3.2.1 Miopía de las Heurísticas Constructivas

Uno de los principales problemas de las heurísticas constructivas es la denominada “miopía”. Ésta consiste en el mal comportamiento de algunas heurísticas, según el cual las decisiones tomadas en cada etapa, aunque buenas según el criterio de selección, perjudican la solución final.

Dos son los modos de solución a este problema: dotar de profundidad a la regla de prioridad o diseñar heurísticas con *BackTracking*.

Dotar de profundidad a la regla de decisión significa que en cada etapa de decisión se tenga en cuenta, en la medida de lo posible, como afectará en las siguientes etapas de decisión la decisión actual. La regla de prioridad en este caso evaluaría no sólo el efecto de la decisión a tomar sino también las decisiones que se tomarían como consecuencia de la que se tome.

Dotar a las heurísticas de técnicas de *BackTracking*. Es decir dotar de capacidad a las heurísticas de volver atrás en las decisiones (y tomar nuevas decisiones) si la situación no es la deseable.

#### 11.3.2.2 Heurísticas con *BackTracking*

Las heurísticas con *BackTracking* buscan una solución probando una de entre varias alternativas, cada decisión se almacena de tal modo que si se puede demostrar que una decisión fue incorrecta se restablezca la solución en ese punto. Conceptualmente un algoritmo de *bracktracking* es equivalente a una búsqueda en profundidad en un algoritmo enumerativo, donde cada decisión equivale a un nodo en el árbol.

#### 11.3.2.3 Procedimientos de enumeración incompletos.

Como se ha dicho, la Optimización Combinatoria es una rama de la Optimización de las matemáticas aplicadas en el cual el espacio de soluciones posibles es discreto. Es decir, el óptimo se podría alcanzar mediante la enumeración de todas las soluciones, aunque esta posibilidad está re*string*ida a tamaños muy pequeños de problema.

Las técnicas de enumeración pueden ser utilizadas para diseñar procedimientos constructivos que sean más rápidos aunque menos eficaces reduciendo el número de nodos a explorar o limitando el tiempo de ejecución.

#### 11.3.2.4 Heurísticas Constructivas. Estructura General y Alternativas de Diseño

Las Heurísticas constructivas se pueden utilizar siempre que el problema se puede resolver mediante decisiones sucesivas (Rutas, Secuenciación, Líneas de Montaje…). En muchas ocasiones el objeto de la decisión es evidente, pero no necesariamente es así.

Se consideran heurísticas voraces aquellas que avanzan sin posibilidad de deshacer las decisiones tomadas.

# 12 ALGORITMOS ENUMERATIVOS

## 12.1 Concepto de Algoritmo Enumerativo

Un algoritmo Enumerativo o de Enumeración Completa exacto es aquel que garantiza la obtención de la mejor solución posible, el óptimo, por la vía de explorar el espacio de soluciones en su totalidad.

En el problema citado para ejemplificar la búsqueda aleatoria (el *Flow-Shop* con 12 tareas) el número de alternativas posibles es de 479.001.600. Si en generar y evaluar cada solución se tardará una milésima de segundo el problema requeriría más de 5 días de computación si se pudiera hacer de un modo ordenado y sistemático (de un modo aleatorio sería sencillamente imposible).

A la vista de las anteriores magnitudes es evidente que los métodos de enumeración completa no son prácticos en la mayor parte de situaciones normales. Sin embargo son una buena estructura para comenzar la búsqueda de soluciones.

Los procedimientos de Enumeración Completa se pueden clasificar en ciegos o guiados según la forma de dirigir la exploración. Según si buscan una solución exacta o se conforman con una buena solución los procedimientos son exactos o heurísticos. También se pueden clasificar según si buscan una solución óptima, todas las soluciones óptimas o todas las factibles.

Por último los algoritmos pueden utilizar la “Fuerza Bruta” analizando una por una todas las soluciones, o pueden tener algún tipo de inteligencia que permita podar aquellas ramas del árbol de exploración que no van a ser útiles.

## 12.2 Algoritmo de Enumeración Completa

Los algoritmos de enumeración completa son diferentes en función de la estructura de la solución. Así por ejemplo no es lo mismo hacer enumeración completa para soluciones que se representan como un vector de *n* dimensiones donde los valores pueden ser 0 o 1, que para una solución que se representa como una permutación de valores, o una solución cuya estructura sea una combinación de *n* elementos tomados de *r* en *r*.

A continuación se presenta un algoritmo de enumeración completa para problemas de permutación basado en una propuesta de Dijkstra citado en http://www.cut-the-knot.org/do_you_know/AllPerm.shtml.

***Ejemplo 19***

```
For c = 1 To M
  i = N - 1
  Do While (S(i - 1) >= S(i)): i = i - 1: Loop
  j = N
  Do While (valor(j - 1) <= valor( i - 1)): j = j - 1: Loop
  Intercambia S( i - 1) con S(j - 1)
  i = i + 1
  j = N
    Do While (i < j)
            Intercambia S( i - 1) con S(j - 1)
      i = i + 1
      j = j - 1
    Loop
```

```
Next c
```

El algoritmo propuesto anteriormente enumera de modo lexicográfico a partir de una solución inicial expresada en el vector S.

El que se expresa a continuación enumera combinaciones (r,n) de la página 231 de Matemáticas discreta Escrito por Richard Johnsonbaugh, Óscar Alfredo Palmas Velasco

```
Salida:  Todas las r-combinaciones de {1,2,. . .,n} en orden lexicográfico creciente.
1.       Procedure combination (r,n)
2.       For i:= 1 to r do
3.       s_i:=i
4.       Print s_1,…,s_r //se imprime la primera r-combinación
5.       For i:= 2 to C(n,r) do
6.       Begin
7.       m:=r
8.       max_val:=n
9.       While s_m = max_val do
10.      //se determina el elemento más a la derecha, que no tenga su máximo valor
11.      Begin
12.      m:=m-1
13.      Max_val := max_val – 1
14.      End
15.      // se incrementa el elemento más a la derecha
16.      s_m:=s_m +1
17.      //el resto de los elementos son los sucesores de s_m
18.      For j.=m+1 to r do
19.      s_i:=s_(i-1) +1
20.      Print s_1,…,s_r //se imprime la i-ésima combinación
21.      End
22.      End combination
```

## *12.3 Estructura de un algoritmo de exploración completa basado en exploración de nodos*

### 12.3.1 Conceptos previos: Nodo

Se puede plantear que los procedimientos de exploración siguiendo un sistema arborescente de nodos que se evalúan, se podan, se seleccionan y se explotan.

Cada nodo hace referencia a una solución parcial (o relajada) del problema. Dicha solución parcial puede ser valorada, por lo que ya se sabe de la solución, o por lo que las futuras incorporaciones a la solución incorporarán.

Cada nodo, excepto el nodo final, es descendiente de algún nodo padre, al que completa parcialmente la solución que aquel traía de sus antecedentes.

Por la vía de la explosión, se generan descendientes a cada nodo se completa un poco más la solución, añadiéndole un sub-conjunto de opciones que permiten definir un poco más la información.

### 12.3.2 Estructura general de un procedimiento de exploración completa basado en nodos

La estructura propuesta parte de la estructura básica nodo que mantiene toda la información relativa a los nodos (como mínimo: cuál es el nodo padre, evaluación, que información incorpora a la solución del nodo padre, si está explotado o no).

Una estructura superior al nodo la constituye la lista de nodos. Dicha lista almacenará todos los nodos abiertos (podría almacenar todos los nodos).

Mientras haya algún nodo abierto el proceso se repetirá con las siguientes 3 fases: seleccionar el siguiente nodo a explotar, explotar el nodo seleccionado (que incluye evaluar los nuevos nodos y cerrar el nodo ya explotado) y una última fase que cierra aquellos nodos que no se tengan, o no se quieran, mantener abiertos. Además cada vez que se genere una solución completa hay que comprobar si mejora la mejor solución disponible hasta este momento.

***Ejemplo 20***

```
GenerarNodoOriginal;
GenerarUnaSolucionInicial; // esto es opcional
While haya_nodos_abiertos do begin
    SeleccionarNodoExplotar;
    ExplotarNodoSeleccionado; // incluye EvaluarNuevosNodos;
    ComprobarSiSeHaAlcanzadoUnaSoluciónMejor;
    CerrarNodosNoÚtiles;
End; //del while
```

A continuación se desarrollan las funciones básicas que hacen falta para el uso de esta estructura que son:

a) La función de evaluación de cada nodo.

b) El modo de seleccionar el nodo a explotar.

c) El modo de explotar el nodo seleccionado.

d) Los criterios por los cuales se cierran nodos

### 12.3.3 Funciones de evaluación

La función de evaluación considera dos paquetes de información que suministra la solución parcial que, como tal, tiene el nodo. Por una lado la parte de la solución que ya está completamente detallada. Por el otro la información asociada a la parte de la solución que no se ha definido.

Según lo dicho, la función de evaluación del nodo, sea *f(n)*, se compone de una parte ligada la solución parcial conocida, sea *g(n)*, y sea *h(n)* la evaluación de la solución que quedaría por definir.

Si el componente de la función de evaluación del nodo ligado a la fase ya definida de la solución se conoce de modo exacto se definirá como *g*(n)*. Si el componente de la función de evaluación del nodo ligado a la parte no definida de la solución se conoce de modo exacto se definirá como *h*(n)*.

Teniendo en cuenta la existencia de estos dos componentes se pueden definir diferentes modos de evaluar la calidad de cada nodo (que servirá posteriormente para compararlo entre ellos). Algunos de los parámetros que permiten crear los diferentes criterios son:

De modo general se puede definir $f(n)=(2-w(n))g(n)+w(n)h(n)$, donde $w(n)$ es un coeficiente que pondera la importancia de $g$ y $h$ en función del nodo en el que se encuentre la exploración.

En general, y salvo que se tenga una razón fundada, se puede asumir que $w(n)=1$ para todo $n$.

Es importante destacar que, en ocasiones, $g(n)$ y $h(n)$ serán muy costosos de calcular. Este hecho requiere una evaluación especial dado que el número de ocasiones en que haya que hacer el cálculo puede ser considerable.

### 12.3.4 Selección del Nodo a explotar

De entre los nodos abiertos hay que elegir cual es el nodo a explotar. Cada criterio tiene sus ventajas e inconvenientes ligadas a:

b) La facilidad de búsqueda del nodo

c) La rapidez en la que se encuentran soluciones completas

d) La rapidez en la que se encuentran buenas soluciones completas

Los criterios más habituales son:

a) Selección aleatoria

b) Primero el más antiguo (FIFO)

c) Primero el más moderno (LIFO)

d) Primero el más profundo (búsqueda en profundidad)

e) Primero el menos profundo (Búsqueda en Anchura)

f) Primero el de menor coste ($g(n)$)

g) Primero el de mejor cota ($f(n)$)

h) Primero el de mejor esperanza

### 12.3.5 Generación de nodos (explosión)

Cada problema tiene un mecanismo de generación de nodos (una vez seleccionado el mismo) diferente. No es posible proponer un mecanismo general. Aunque sí hay algunos criterios que se deben observar. En primer lugar debe generar todas las opciones posibles a partir del nodo explotado, y no debe conducir a la generación de una cantidad excesiva de nodos en cada explosión.

En los problemas en que la solución es una permutación de valores un modo habitual de explotar nodos es añadir un nuevo valor a la solución parcialmente construida.

### 12.3.6 Eliminación (o cierre) de Nodos.

No todos los nodos abiertos son igualmente necesarios, por ello se suele añadir un procedimiento que elimine aquellos nodos que no se quieran mantener abiertos.

Algunos de los nodos abiertos serán manifiestamente incompatibles y por ello deben ser cerrados. Otros nodos serán estructuralmente iguales que otros nodos ya creados (abiertos o no), podría interesar buscar esos nodos para cerrar el nodo idéntico.

Es posible que haya nodos que conduzcan de manera unívoca a la generación de una solución completa. En ese caso hay que crearla y valorarla, comparándola con la mejor solución disponible hasta el momento, guardándola si es mejor que ella y en cualquier caso cerrándolas.

Algunos nodos estarán dominados por otros nodos ya creados, en ese caso se debe cerrar el nodo. Otros nodos podrían tener alguna característica que les impidiera conducir a un óptimo en cuyo caso pueden también ser cerrados.

En algunos casos la cota de los nodos abiertos es ya peor que la mejor de las soluciones obtenidas. En ese caso no tiene sentido seguir explotando ese nodo y conviene que sea cerrado.

Por último, es posible que por problemas de memoria no se pueda tener abierta más de una cierta cantidad de nodos. En ese caso hay que elegir el conjunto de nodos a mantener y cerrar el resto.

Los procedimientos de enumeración completa analizan todos y cada uno de los nodos que se abren.

## *12.4 Otras técnicas de Enumeración basadas en la exploración por nodos*

Las técnicas de Enumeración Implícita analizan que nodos no se deben abrir puesto que se puede deducir de su estructura los resultados que puede obtener.

Las técnicas *Beam Search* avanzan por niveles y no permiten que haya más de *M* nodos abiertos simultáneamente.

Las técnicas *Laser* generan, a partir de cada nodo abierto, soluciones, mediante heurísticas rápidas, aunque la exploración se hace también por niveles. Este método permite tener buenas soluciones sin necesidad de acabar el procedimiento que garantizaría el óptimo.

# 13 PROCEDIMIENTOS DE MEJORA LOCAL

Son procedimientos basados en analizar el vecindario de una determinada solución, para averiguar si existe, en dicho vecindario, una solución con mejor valor de función objetivo.

El proceso de búsqueda examina todas las opciones dentro del vecindario actual y selecciona (generalmente) la mejor opción para moverse hacia él y recomenzar el proceso

## 13.1 Vecindario

La definición de vecindario es el conjunto de soluciones que pueden ser alcanzados desde la solución actual con una operación simple. Cualquier punto que requiera dos operaciones no es vecindario.

La mejor solución en un vecindario es un óptimo con respecto a su vecindario.

A continuación se describen los tres tipos de vecindario más habituales.

### 13.1.1 Intercambio de 2 elementos

Son vecinos de una determinada solución todas aquellas soluciones alcanzables mediante el intercambio de dos elementos de la solución.

Ejemplo: Sea ABCDEFGH la solución original considerada. Un intercambio de los elementos 3 y 6 proporcionaría la siguiente solución ABFDECGH.

### 13.1.2 2-opt

Son vecinos de una determinada solución todas aquellas soluciones mediante el siguiente movimiento: Tras seleccionar dos elementos de una solución invertir el tramo que hay entre los mismos. Este tipo de vecindario ha demostrado su superioridad, en general, frente al del intercambio simple. Además todos los vecinos según el proceso de intercambio son vecinos en segundo nivel del vecindario 2-opt. Lo mismo ocurre con los vecinos encontrados mediante el proceso Insertar.

Ejemplo: Sea ABCDEFGH la solución original considerada. Un 2-opt de los elementos 3 y 6 proporcionaría la siguiente solución ABFEDCGH.

### 13.1.3 Insertar

Son vecinos de una solución mediante el proceso de inserción aquellos en los que se coge un elemento y se inserta entre otros dos elementos. Los vecinos así conseguidos son vecinos en segundo nivel del vecindario 2-opt.

Ejemplo: Sea ABCDEFGH la solución original considerada. Insertar 3 en 6 proporcionaría la siguiente solución ABDEFCGH.

## *13.2 Algoritmos de Mejora basados en Vecindario*

### 13.2.1 Nomenclatura

*x: solución actual*
*x': solución en el vecindario*
*c(x) = valor de la función objetivo para x*
*N(x): vecindario de x*
*random: función*

### 13.2.2 Mejora Iterativa Simple. Procedimiento

*Paso 1) Inicialización*
*1.1 Seleccionar una solución inicial x*
*1.2 Definir mejor_coste:=c(x)*
*Paso 2) Selección (y acabar si no hay mejora)*
*2.1 Seleccionar x' perteneciente N(x) tal que c(x')<c(x)*
*2.2 Si no hay tal x' entonces STOP*
*Paso 3) Actualizar*
*3.1 x:=x'*
*3.2 mejor_coste:=c(x)*
*3.3 Ir a paso 2.*

### 13.2.3 Descenso Rápido. Procedimiento

*Paso 1) Inicialización*
*1.1 Seleccionar una solución inicial x*
*1.2 Definir mejor_coste:=c(x)*
*Paso 2) Selección (y acabar si no hay mejora)*
*2.1 Seleccionar x' perteneciente N(x) tal que c(x')<c(x) y es el menor coste en el vecindario c(x')=min(N(x))*
*2.2 Si no hay tal x' entonces STOP*
*Paso 3) Actualizar*
*3.1 x:=x'*
*3.2 mejor_coste:=c(x)*
*3.3 Ir a paso 2.*

### 13.2.4 Mejora Iterativa Aleatorizada. Procedimiento

*wp: real positivo menor que 1.*
*Paso 1) Inicialización*
*1.1 Seleccionar una solución inicial x*
*1.2 Definir mejor_coste:=c(x)*
*Paso 2) Selección*
*2.1 Si random<wp seleccionar x' perteneciente N(x) de modo aleatorio*
*2.2 en caso contrario Seleccionar x' perteneciente N(x) tal que c(x')<c(x)*
*Paso 3) Actualizar*
*3.1 x:=x'*
*3.2 mejor_coste:=c(x)*
*3.3 Mientras no se produzca la condición de terminación ir a paso 2.*

# 14 PROCEDIMIENTOS META-HEURÍSTICOS

Los procedimientos meta-heurísticos son un marco de referencia de alto nivel que se especializa para resolver problemas de optimización. Es también una estrategia que guía otras heurísticas en la búsqueda de mejores soluciones factibles.

Las meta-heurísticas deberían ser generales y simples. A medida que se especializan pierden la generalidad y exigen más conocimiento del problema específico.

Además de los parámetros que definen cada algoritmo específico, las meta-heurísticas requieren algún tipo de representación que por tanto es dependiente de cada problema.

Entre las meta-heurísticas más conocidas se puede destacar:

- Algoritmos Genéticos
- *Scatter Search*
- Algoritmos Meméticos
- Algoritmos de Hormigas
- Redes Neuronales
- *GRASP*
- Recocido Simulado
- Búsqueda Tabú
- *Iterated Local Search*, *Variable Neighborhood Search*

## 14.1 Procedimientos de población

Según (Osman, 1995) las meta-heurísticas de población, a la que pertenecen los algoritmos genéticos, presentan todo o parte del siguiente esquema de la Ilustración 1: Estructura genérica de las Meta-heurísticas (Osman, 1995). En este esquema se puede observar que todos los procedimientos empiezan obteniendo una generación inicial (que algunas veces pueden estar constituidas por un solo individuo). Esta generación pasa a una segunda etapa (búsqueda por generación de movimientos) o a una tercera (búsqueda por generación de soluciones). En la segunda etapa puede ser usado cualquier procedimiento de búsqueda local que generan vecindarios a partir de una solución única. Los métodos de búsqueda por individuos pueden operar sobre cada individuo de la generación actual.

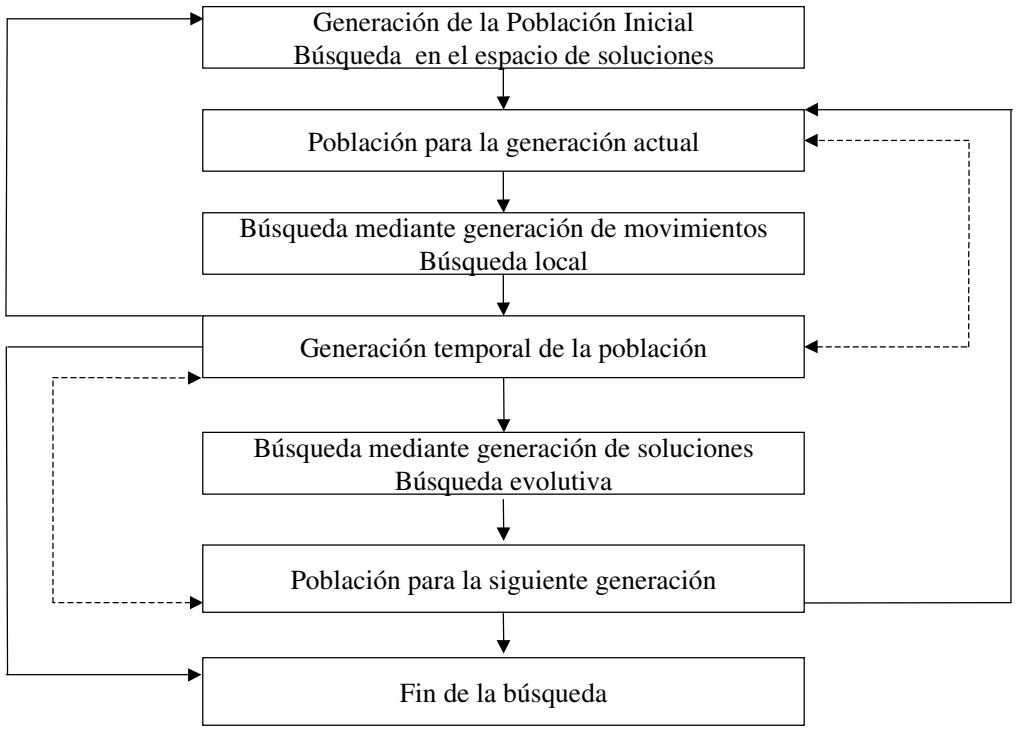



*Ilustración 1: Estructura genérica de las Meta-heurísticas (Osman, 1995)*

Al final de la segunda etapa puede finalizar la búsqueda en la población actual e ir a la etapa cuatro o reinicializada volviendo a la etapa primera si la búsqueda evolutiva no se incluye, o la población actual pasa por la etapa tercera. En esta tercera etapa se realiza una búsqueda basada en la generación de nuevas soluciones. Al final de esta etapa se puede volver a la segunda o repetir otro ciclo de búsqueda evolutiva. Las líneas punteadas indican opciones que pueden ser utilizadas o no.

### 14.1.1 Algoritmos Genéticos

Los Algoritmos Genéticos fueron introducidos por Holland en 1975 para imitar algunos de los mecanismos que se observan en la evolución de las especies. Basándose en las principales características de la reproducción sexual, Holland creó un algoritmo que genera nuevas soluciones a partir de la unión de soluciones progenitoras utilizando operadores similares a los de la reproducción, sin necesidad de conocer el tipo de problema a resolver.

Una ventaja importante que presentan las heurísticas frente a las técnicas que buscan soluciones exactas es que, por lo general, permiten una mayor flexibilidad para el manejo de las características del problema. No suele ser complejo utilizar algoritmos heurísticos que en lugar de funciones lineales utilicen no-linealidades. Habitualmente las heurísticas proponen un conjunto de soluciones, ampliando de esta forma las posibilidades de elección del decisor, especialmente cuando existen factores no cuantificables que no han podido ser reflejados en el modelo pero deben ser tenidos en cuenta.

En resumen, podría decirse que el uso de estas técnicas supone la posibilidad de resolver, de forma práctica, problemas de gran complejidad que resultaban intratables mediante técnicas exactas y permite definir con detalle el metabolismo del sistema.

Los algoritmos genéticos son una clase de algoritmos inspirados en los mecanismos de la genética, que se aplican a problemas de optimización (especialmente a los problemas de combinatoria).

"Procedimiento basado en la analogía con la evolución de los seres vivos. La premisa que subyace a este tipo de enfoques es, que se puede encontrar soluciones aproximadas a problemas de gran complejidad computacional mediante un procedimiento de evolución simulada matemáticamente en un ordenador " [Holland, 1975]

Un algoritmo genético es una meta-heurística que permite, a partir de una población inicial de soluciones, obtener soluciones potencialmente mejores mediante el cruce de varias de las soluciones originales.

Requieren el diseño de tres operadores (generalmente de tipo probabilístico) que actúan sobre objetos denominados "*strings*".

- Reproducción. Selección de *strings* para poder proceder a la reproducción
- Cruce. Combinación de dos o más *strings* para que intercambien valores, reproduciéndose.
- Mutación. Alteración espontánea de los elementos de un *string*

Estos procesos pueden tener cada uno formas variadas, e incluso se permite el avance en paralelo de algoritmos genéticos con el mismo propósito.

#### 14.1.1.1 Estructura General de un Algoritmo Genético.

*Generación de la Población Inicial*
*Repetir*
    *Elegir dos (o más) padres de entre la población.*
    *Construir una solución nueva a partir de la estructura de los padres elegidos.*
    *Si se considera conveniente provocar mutaciones.*
    *Decidir si la nueva solución pasa a formar parte de la población.*
*Hasta alcanzar un criterio de parada*

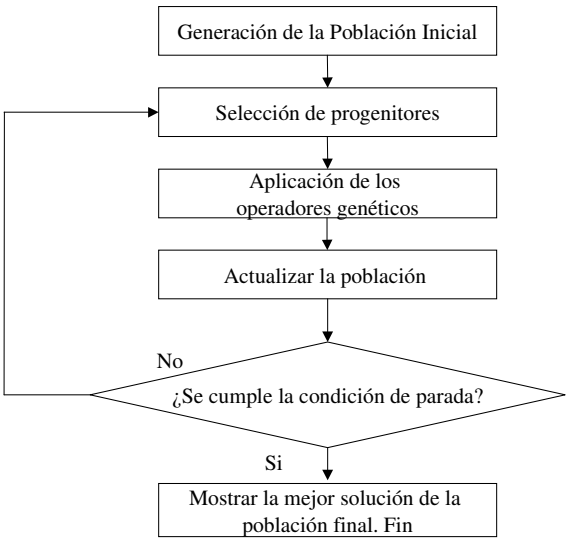



Figura X

#### 14.1.1.2 Representación de las soluciones (representación cromosómica)

Inicialmente en los algoritmos genéticos se representaba la solución mediante vectores de carácter binario. Los operadores eran muy sencillos pero muchas de las soluciones obtenidas no eran factibles con lo cual posteriormente había que reparar.

Otro modo de representar soluciones útiles para los algoritmos genéticos es utilizar un vector unidimensional de elementos independientes. (Por ejemplo un vector para el Problema TSP).

Cuando el problema es más complejo (por ejemplo un problema de *Flow Shop* Híbrido), se puede exigir *N* Vectores con codificación combinada. En el caso del *Flow Shop* Híbrido un buen modo de representar es un vector que representa la secuencia de operaciones y otro vector las máquinas donde cada producto va destinado.

También se pueden utilizar vectores n-dimensionales cuyo problema fundamental es el de definir métodos de cruce eficientes (Ej: Vector Bidimensional para la Programación de Producción en Máquinas en Paralelo).

Por último existen vectores con codificación implícita como por ejemplo para la codificación del balance de líneas en los que una correcta codificación es una exigencia para obtener resultados razonables.

#### 14.1.1.3 Definición de la Medida de Evaluación de cada Solución (*Fitness*)

El procedimiento de evaluación de una solución es especialmente importante en un algoritmo genético, pues se va a utilizar muchas veces.

- Fitness(x)= Valor función objetivo
- Fitness(x)= Valor función objetivo normalizada
- Fitness(x)= Valor función objetivo normalizada –Penalización por infactibilidad
- Otros …

#### 14.1.1.4 Definición de Métodos para la Generación de la Población inicial

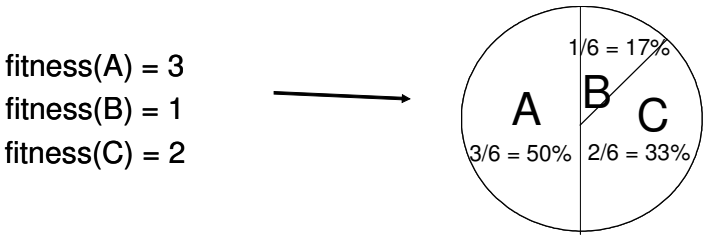
Procedimientos:

- Generación Aleatoria
- Heurísticas conocidas

El tamaño de la Población es especialmente relevante. Un tamaño demasiado grande genera dispersión de resultados Un tamaño demasiado pequeño genera una excesivamente rápida convergencia de soluciones. Una pregunta interesante al diseñar un algoritmo genético es si debe mantenerse constante a lo largo del proceso.

#### 14.1.1.5 Definición de Métodos para la selección de dos (o más) padres de la Población Inicial.

El número de padres seleccionados es generalmente dos (aunque puede ser mayor si el procedimiento de cruce lo permite). El modo de selección de los padres se suele realizar mediante el uso de la denominada ruleta. Se asignan probabilidades a los padres en función de su *fitness*.




14.1.1.6 Operaciones de Recombinación o Cruce de los individuos seleccionados

El cruce o intercambio de características entre las soluciones elegidas produce un fenómeno de convergencia puesto que hace que los individuos de la población cada vez se parezca más entre sí.

Existen infinidad de operadores de cruce, los más sencillos son los desarrollados para representaciones binarias como el "*one-point crossover*" que intercambia a partir de una posición determinada la cadena binaria que representa el individuo de cada padre. Este concepto se ha refinado con el "*two-point crossover*" que intercambia la parte del individuo que hay entre dos posiciones definidas de manera aleatoria.

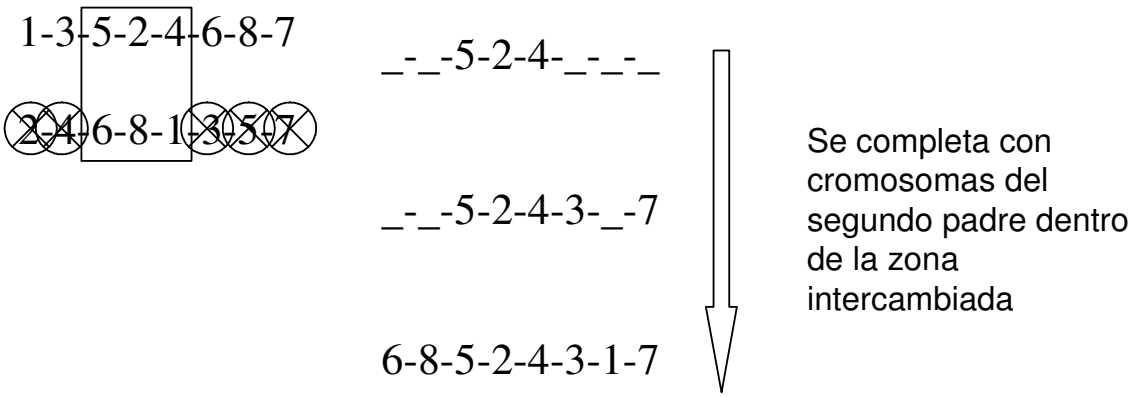



Figura X

14.1.1.7 Generación Soluciones Iniciales

Para generar las soluciones iniciales se puede optar por generarlos de manera aleatoria o utilizando heurísticas conocidas que generan buenas soluciones de partida.

El problema de la primera aproximación es que la convergencia a una buena solución puede ser lenta. El problema de la segunda aproximación es que tiene una tendencia inicial que será difícilmente superada.

14.1.1.8 Selección de "Padres"

El número de padres seleccionados es generalmente dos (aunque puede ser mayor si el procedimiento de cruce lo permite).

El modo de selección de los padres se suele realizar mediante el uso de la denominada ruleta. En ella se asigna una probabilidad a cada padre proporcional a su *fitness* de tal modo que la suma de probabilidades sea la unidad.

#### 14.1.1.9 Cruce

El cruce o intercambio de características entre las soluciones elegidas produce un fenómeno de convergencia puesto que hace que los individuos de la población cada vez se parezca más entre sí.

En el caso del operador "*two-point crossover*", en primer lugar, se calculan dos números aleatorios entre 1 y el número de elementos del vector. El operador obtiene del vector "progenitor a" los elementos contenidos entre el primer número y el segundo número. Posición a posición de entre las aún no asignadas se copian los elementos del "progenitor b" si no están incluidos en el fragmento de vector copiado del padre. Los huecos que queden se rellenan con el orden en el que aparecen los valores no incorporados del "progenitor b".

En muchas ocasiones el resultado así obtenido no es una solución factible. En estos casos hay que proceder a realizar una operación de "reparación" en la que se genera factibilidad al resultado obtenido.

#### 14.1.1.10 Actualizar Población

El argumento fundamental de los algoritmos genéticos no es que obtienen buenos resultados a partir de la exploración ligada a una población inicial, sino que la población evoluciona a medida que se obtienen soluciones mejores.

Dicha evolución se consigue por la vía de sustituir (de uno en uno o en paquetes) elementos de la población anterior por soluciones obtenidas durante la ejecución del algoritmo.

#### 14.1.1.11 Monitorización de Convergencia y Mutación

Cada cierto tiempo es interesante realizar operaciones de mutación, para garantizar que la población no converja demasiado rápidamente. Generalmente son operaciones en las que el vector obtenido es afectado por alguna operación de búsqueda de vecindario como la inserción o el intercambio de dos elementos.

La monitorización de la convergencia permite lanzar procedimientos de mutación si es necesario. Un modo de monitorizar la convergencia es contabilizar para toda la población si se repiten los mismos valores en las posiciones.

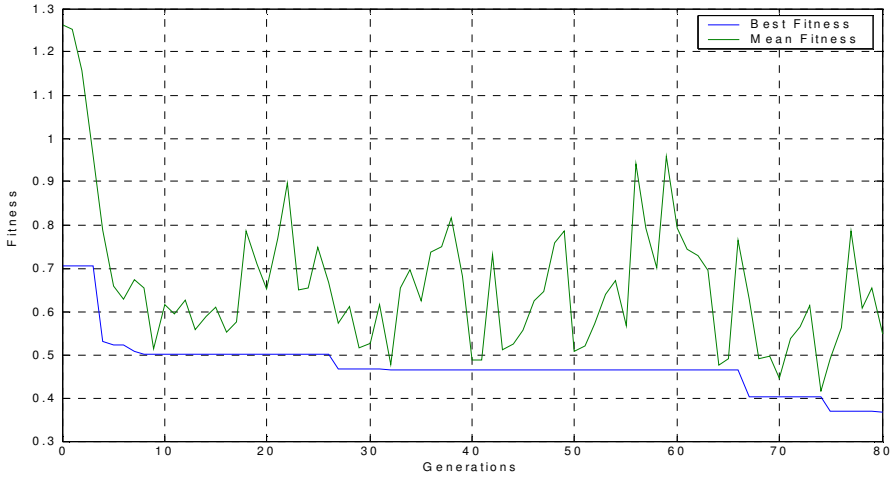


Figura X

Puede ser interesante aplicar cada cierto tiempo pequeñas mutaciones a las soluciones obtenidas, ya sea para mejorar la población, ya sea para evitar la excesiva convergencia en la población. Los tipos básicos de mutación recuerdan a los vecindarios definidos en otro lugar de estos apuntes (2-opt, Inversión, Inserción) pero también podría ser la introducción selectiva de elementos "ajenos" a la evolución de la solución.

### 14.1.2 *Scatter Search* y Algoritmos Meméticos

La *Scatter Search* o búsqueda dispersa es un método meta-heurístico de población para resolver problemas de optimización. Aunque fue originalmente introducido en los años setenta, recientemente es cuando ha sido probado en numerosos problemas difíciles con gran éxito (Glover, Laguna, and Marti, 2003).

Pertenece a la familia de los llamados Algoritmos Evolutivos, y aunque presenta similitudes con los Algoritmos Genéticos, difiere de éstos en principios fundamentales, tales como el uso de estrategias sistemáticas en lugar de aleatorias o la posibilidad de combinar más de dos elementos.

*Scatter Search* proporciona un marco flexible que permite el desarrollo de diferentes implementaciones con distintos grados de complejidad. El método consta de las siguientes etapas: Generación de la Diversidad, Método de Mejora sobre la población anterior, selección de la población de referencia, selección de un sub-conjunto para la combinación, combinación y actualización de la población de referencia.

Comienza con una colección de puntos de referencia (obtenidos mediante la aplicación de otros procedimientos). Se parece a los algoritmos genéticos excepto que en la búsqueda "scatter" la combinación de elementos de la población es una combinación lineal que es posteriormente adaptada para convertirse en una solución.

Los algoritmos meméticos también pertenecen a los algoritmos basados en poblaciones. Explicado de una manera muy básica utiliza el operador de cruce de los algoritmos genéticos, pero al obtener un "hijo" le aplica métodos de mejora local.

## *14.2 Meta-heurísticas de Búsqueda de Entornos*

### 14.2.1 *GRASP*

Probablemente el modo más sencillo para evitar los óptimos locales es empezar por puntos diferentes cada vez el camino de descenso(Marti, 2003). Esta es la lógica básica del procedimiento denominado *Greedy* Randomized Adaptive *Search* Procedure-*GRASP*- (Resende and Ribeiro, 2003). La generación de los puntos de inicio es la parte voraz, aleatoria y adaptativa del proceso mientras que la búsqueda local es el componente *search*. Este algoritmo de muy fácil implementación, puede ser también paralelizado o integrado con otras meta-heurísticas.

*GRASP* es el acrónimo de *Greedy* Randomized Adaptive *Search* Procedures. Es una meta-heurística que empieza la resolución del problema en diferentes puntos del espacio de soluciones y una vez "re-empieza" utiliza un procedimiento heurístico para completar o mejorar la solución, alcanzando un óptimo local.

La definición de la situación inicial puede ser más o menos aleatoria. El método de resolución puede ser una heurística específica del problema u otra meta-heurística de búsqueda local.

El modo más sencillo para evitar los óptimos locales es empezar por puntos diferentes cada vez el camino de descenso (Marti, 2003). Esta es la lógica básica del procedimiento denominado *Greedy Randomized Adaptive Search Procedure-GRASP-* (Resende *et al.*, 2003). La generación de los puntos de inicio es la parte voraz, aleatoria y adaptativa del proceso mientras que la búsqueda local es el componente *search*. Este algoritmo de muy fácil implementación, puede ser también paralelizado o integrado con otras meta-heurísticas.

Otro modo de concebir las estructuras *GRASP* es mediante el uso de heurísticas constructivas aleatorias (*randomizadas*).

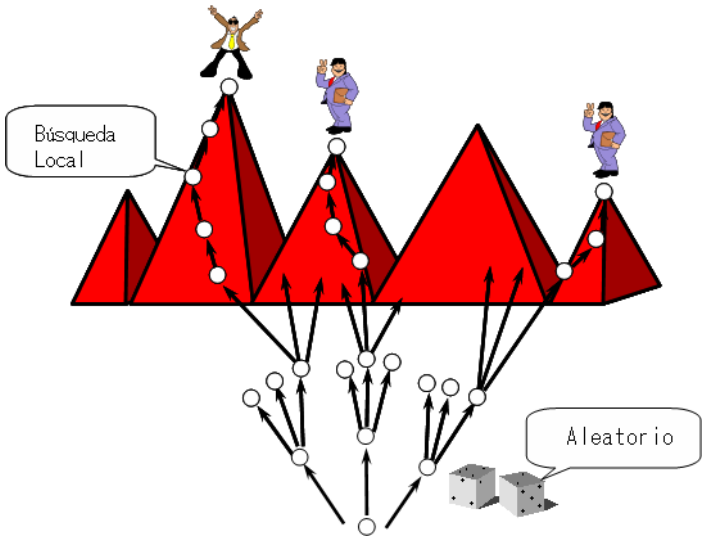



Figura X

Dicha estructura utilizaría un algoritmo heurístico constructivo voraz, sustituyendo el criterio de selección del siguiente elemento a incorporar a la selección por un criterio estocástico. De este modo, en el problema TSP, en lugar de elegir, por ejemplo, el vecino más cercano, se elige aleatoriamente de entre los 3 vecinos más cercanos, o de entre aquellos vecinos que sólo estén un 10% más lejos que el más cercano de los vecinos.

### 14.2.2 ILS y VNS

La idea de reiniciar la búsqueda de nuevo es interesante, pero para espacios de solución extensos puede llevar a un procedimiento no eficiente. Por ello la *Iterated Local Search* (Lourenço *et al.*, 2003) pretende comenzar la búsqueda de nuevo no "por cualquier lado" si no por puntos del espacio que son

modificaciones –no demasiado fuertes- de soluciones ya encontradas previamente. Específicamente se puede citar la Búsqueda de Vecindario Variable (*Variable Neighborhood Search*- VNS), cuya idea fundamental es la búsqueda de mejores soluciones cambiando de vecindario cuando el que se está considerando no va a dar mejores resultados (Hansen *et al.*, 2003). La VNS se comporta como un método de descenso hacia mínimos locales, explorando sistemáticamente o aleatoriamente, cada vez más lejanos vecindarios de la solución. El método salta de la solución actual a una mejor sí y sólo sí encuentra una mejor solución (Perez *et al.*, 2006).

### 14.2.3 Recocido Simulado

Los algoritmos de Recocido Simulado (*Simulated Annealing*) fueron introducidos por Cerny y Kirkpatrick (cita) a principio de la década de los 80 del siglo XX para la optimización de problemas combinatorios con mínimos locales. Utilizan técnicas de optimización no determinista: no buscan la mejor solución en el entorno de la solución actual sino que generan aleatoriamente una solución cercana y la aceptan como la mejor si tiene menor coste, o en caso contrario con una cierta probabilidad *p*; esta probabilidad de aceptación irá disminuyendo con el número de iteraciones y está relacionada con el empeoramiento del coste (Henderson *et al.*, 2003).

El recocido simulado está basado en un cómo funciona el proceso físico de recocido. En el proceso de búsqueda de soluciones permite movimientos que no mejoran con una probabilidad que decrece a lo largo del tiempo. El ratio de dicho decrecimiento queda determinado por el programa de enfriamiento que casi siempre es un parámetro usado como ratio de decrecimiento exponencial.

Ejemplo

```
x: solución actual
c(x) = valor de la función objetivo para x
N(x): vecindario de x

Paso 1) Inicialización
    1.1 Seleccionar una solución inicial x
    1.2 Definir mejor_coste:=c(x)
    1.3 Determinar la T inicial
Paso 2) Mientras no se produzca la condición de terminación
    2.1 Elegir un x' perteneciente N(x)
    2.2 Si x' es aceptable dependiendo del criterio de Recocido Simulado
            x:=x'
    2.3 Actualizar T de acuerdo a la planificación de Recocido Simulado
```

### *14.2.4* Búsqueda Tabú (*Tabu search)*

Otro método meta-heurístico que pretende dotar de "inteligencia" a los algoritmos de búsqueda local es denominado de Búsqueda Tabú (Gendreau, 2003). La búsqueda tabú, a diferencia de otros algoritmos basados en técnicas aleatorias de búsqueda en vecindario, se caracteriza porque utiliza una estrategia basada en el uso de estructuras de memoria para escapar de los óptimos locales, en los que se puede caer al "moverse" de una solución a otra por el espacio de soluciones. Al igual que en la búsqueda local, la búsqueda tabú selecciona de modo agresivo el mejor de los movimientos posibles en

cada paso. Al contrario que sucede en la búsqueda local, se permiten movimientos a soluciones del entorno aunque se produzca un empeoramiento de la función objetivo, de manera que sea posible escapar de los óptimos locales y continuar estratégicamente la búsqueda de mejores soluciones.

La búsqueda tabú es una meta-heurística que resuelve problemas de optimización, basándose en que se prohíbe al sistema deshacer algunos de los últimos movimientos realizados, para permitir la búsqueda por entornos que de otro modo no se explorarían.

Esta meta-heurística resuelve problemas de optimización, basándose en una gestión de memoria multinivel y exploración. Utiliza el concepto de vecindario para probar soluciones. En su versión más simple un procedimiento de búsqueda tabú es como sigue:

```
x: solución actual
c(x) = valor de la función objetivo para x
N(x): vecindario de x

    x:=determinarSoluciónInicial;
    while not(hayQueTerminar) do begin
        V:=conjuntoDeVecinosNoProhibidos(x);
        x':=elegirMejorVecino(V);
        actualizarListaDeMovimientosProhibidos(x,x');
        x:=x';
    end;
```

Hay muchas variaciones, como por ejemplo los niveles de aspiración, que se pueden incluir en especificaciones más complejas.

## *14.3 Meta-heurísticas basadas en el reconocimiento de patrones*

### 14.3.1 Redes Neuronales

Las redes neuronales artificiales, son sistemas de procesamiento de la información cuya estructura y funcionamiento emulan a las redes neuronales biológicas. Los elementos simples de procesamiento de una red neuronal artificial también se denominan neuronas y el peso de sus conexiones, pesos sinápticos. Las redes neuronales tienen una gran utilidad en sistemas de predicción. Mediante un histórico de datos (y de salidas) se establecen las relaciones entre los datos y las salidas (entrenamiento), de tal modo que a un nuevo juego de datos le correspondería una nueva salida.

### 14.3.2 Algoritmo de Colonia de Hormigas (ACO)

Un procedimiento heurístico de búsqueda basado en el comportamiento de las colonias de hormigas. Particularmente su habilidad para encontrar el camino más corte de modo colectivo.

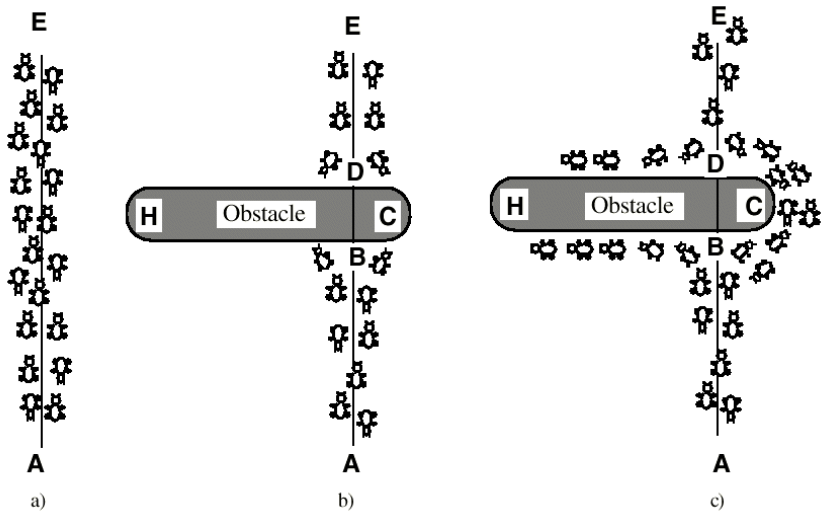



Figura X

Las hormigas tienen memoria que afecta al proceso de construcción del camino de modo que aquel camino que es de menor camino es el que tiene más probabilidad de ser elegido. En los algoritmos ACO se guarda memoria de las conexiones más habituales entre dos elementos consecutivos.

# 15 ALGUNOS EJERCICIOS DE PROGRAMACION MATEMÁTICA

## 15.1 ¿Cuánto gana la Empresa?

Una empresa fabrica 2 productos P y Q.

P se vende a 90 € y Q a 100 €. La demanda de cada producto es de P=100 unidades/semana y de Q=50 unidades/semana.

Los dos productos requieren de una misma pieza central, la materia prima de la cual vale a 20€ la unidad. Para fabricar la pieza central hacen falta 15 minutos del recurso B y 5 minutos del recurso C. Para fabricar el componente 1 del producto P hace falta materia prima por valor de 20€/unidad, 15 minutos del recurso A y 10 minutos del recurso C. Al ensamblar la pieza central con el componente 1 utilizamos otro componente 3 que se compra al precio de 5€/unidad, lo ensambla el recurso D en 15 minutos cada unidad.

El producto Q sigue un procedimiento similar. El componente 2 utiliza Materia Prima por valor de 20€/unidad, pasa por el recurso A donde está 10 minutos y luego por el proceso B donde está 15 minutos. Finalmente es ensamblado por el recurso D en 5 minutos. El mes tiene 20 días de 8 horas. Los gastos totales son 3600€/semana.

a) ¿Cuál es el mejor plan de producción para la empresa? ¿Qué beneficio le aporta?
b) ¿Cuál es el valor de una hora más de cada recurso productivo?
c) Establecer un objetivo que pretenda maximizar el ratio beneficio entre horas totales de trabajo.
d) ¿Cómo incorporar limitaciones en la disponibilidad de materia prima?
e) ¿Cómo incorporar un número indefinido de productos al modelo?

## 15.2 Problema de Corte

Una empresa tiene barras metálicas de 12,5 metros. Un día cualquiera recibe pedidos que acumulan los siguientes cortes: 5 barras de 3 metros, 4 barras de 4 metros, 3 barras de 5 metros, 5 barras de 6,5 metros.

Defina un modelo que genere un plan de corte con el mínimo resto posible.

## 15.3 Centralita Telefónica

Una empresa tiene una centralita telefónica que debe atender durante 17 horas al día. La demanda de operadores para las diferentes horas es variable. Puede contratar trabajadores a una ETT por 4,5,6,7 y 8 horas al día. Cada tipo de contrato tiene un coste diferente.

Defina cuántos contratos de cada tipo ha de hacer para cada hora de una jornada.

## *15.4 Varios Turnos*

Una fábrica puede trabajar a 3 turnos, con costes fijos por turno de (1000,500,200) y costes variables diferentes en cada turno por unidad (20,30,40). Las capacidades productivas son también diferentes (100,800,700). El precio de venta de cada unidad es de 50.

a) ¿Cuántas unidades debe fabricar?
b) Si variara el precio con la cantidad puesta en el mercado según una recta con los siguientes puntos extremos: si vende a 100 el mercado demanda 500. Y si vende a 50 el mercado demanda 3000. ¿Qué precio debe imponer?

## *15.5 Plan de Producción*

Sea una empresa que fabrica 3 tipos de productos. Para ello utiliza 2 tipos de máquinas (Mecanizado y Acabado). Tiene 3 máquinas de Mecanizado y 2 de Acabado.

Cada centenar de producto A requiere 2 horas de Mecanizado y 2 horas de Acabado.

Cada centenar de producto B requiere 3 horas de Mecanizado y 2 horas de Acabado.

Cada centenar de producto C requiere 2 horas de Mecanizado y 1 horas de Acabado.

Además para poner en marcha cada máquina hay que utilizar mano de obra, de la que disponemos 4 unidades por turno. Los costes de personal semanales se evalúan en 3000€.

La semana laboral tiene 120 horas.

Se plantean dos escenarios de precios por centenar de unidades con probabilidades respectivas del 30% y del 70%.

| | Prob. | Valor A | Valor B | Valor C |
|---|---|---|---|---|
| e=1 | 30% | 50 | 50 | 40 |
| e=2 | 70% | 40 | 60 | 30 |

Las demandas están limitadas en 5000 unidades de producto A y 4000 unidades de producto B y 5000 unidades de producto C.

1. Plantear el problema para el beneficio promedio esperado.
2. Plantear el problema para Maximizar el mínimo beneficio con Programación Lineal.
3. Maximizar el beneficio por hora trabajada con Programación Lineal.
4. Plantear el problema para maximizar el uso de la mano de obra, el beneficio promedio y el mínimo beneficio.
5. Relajar la restricción de mano de obra con escenario=1. El coste de la hora extra son 5€. No se hacen horas extra en fin de semana.

6. Imponer un consumo mínimo de 320 horas de mano de obra con el menor número de desigualdades posible.
7. Plantear el dual del problema original.
8. Suponer que en el escenario 1 existe una relación entre el precio y la cantidad vendida, de tal modo que el producto entre la cantidad vendida y la raíz cuadrada del precio de venta es constante e igual a 63.
9. Suponer que se puede alquilar una máquina más de mecanizado con un coste fijo semanal de 300€ y variable de 2€ por hora utilizada
10. Suponer que se puede alquilar una máquina más de mecanizado con un coste fijo semanal de 300€ y variable de 2€ por unidad fabricada.

## *15.6 Localización*

Dado un conjunto de ciudades, con un beneficio asociado de servirlas y un coste de implantar un centro de servicio en cada una de ellas, y unidas mediante carreteras de diferente longitud grafo conexo). Asumiendo que desde una ciudad se puede servir a otras ciudades si la distancia no es superior a un cierto valor.

a) ¿Cómo cubrir todas las ciudades a mínimo coste?
b) ¿Cómo obtener el máximo beneficio con coste limitado?
c) ¿Cómo obtener máxima rentabilidad?
d) ¿Cómo imponer que se debe implantar en A o en B?
e) ¿Cómo imponer que se debe implantar en A y en B?
f) ¿Cómo imponer que si implantas en A debes hacerlo en B?
g) ¿Cómo imponer que si implantas en A no debes hacerlo en B?
h) ¿Cómo imponer que si implantas en A y en B debes hacerlo en C?

## *15.7 Vinos "Don Pepón"*

"Don Pepón" es la marca comercial de una empresa dedicada a la elaboración de vinos, zumos y licores. Cuando llega la época de la vendimia, se debe decidir qué cantidad de 2 tipos básicos de uvas debe comprar.

La uva la utiliza para fermentarla, tras un proceso de mezcla directamente dando lugar a tres tipos de vinos (Tinto, Rosado y Blanco). También la puede utilizar para venderla como zumo de uva. Y por último la puede fermentar de un modo acelerado y tras destilarla obtener alcohol etílico y un poso que se utiliza también para el zumo de uva.

Cada kilo de uva comprado da lugar a 0,80 litros de cualquiera de los líquidos finales.

El vino Tinto debe tener un porcentaje máximo de 15% de uva de tipo 2. El vino Rosado debe tener un porcentaje máximo de 35% de uva de tipo 2 y un porcentaje mínimo de 40% de uva de tipo 1. El vino Blanco debe tener un porcentaje máximo de 15% de uva de tipo 1.

El zumo de uva se produce con un mínimo de un 70% en volumen de uva, el resto es agua. Aunque no se debe incorporar más de un 15% de poso de uva de tipo 1, o más de un 30% de poso de uva de tipo 2.

Por cada tonelada de uva de tipo 1 se generan 1,6 hectólitros de alcohol etílico y 2,4 hectólitros de poso mediante la fermentación acelerada, y por cada tonelada de uva de tipo 2 se generan 1,2 hectólitros de alcohol etílico y 2 hectólitros de poso.

El precio en el mercado mayorista del vino tinto “Don Pepón” es de 1 € el litro, el vino Rosado se vende a 0,9 € por litro, y el vino blanco a 0,8 € por litro. El litro de zumo de uva está a 0,5 € por litro. Mientras que el alcohol etílico obtenido de la uva se vende a 1,2 € por litro.

Los precios de compra actuales de cada tipo de uva son 60 € los 100 kilos de tipo 1, y 50€ los 100 kilos de uva de tipo 2.

Sean X1 y X2 las toneladas de uva de cada tipo que se compran. Sean V1, V2 y V3 los hectolitros de cada tipo de vino que se venden. Sea Z los hectolitros de zumo e Y los hectolitros de alcohol etílico.

a) Fija una función objetivo para un modelo que pretenda maximizar beneficios.

b) Fija las restricciones del modelo anterior.

c) Incorpora la restricción de que la capacidad de fermentación máxima de vino es de 5000 hectolitros.

d) Incorpora las restricciones financieras según las cuales no es posible adquirir más 600.000 € de uva en esta temporada.

e) Suponga que algunas limitaciones del mercado imponen que al menos un 20% del Vino total producido debe ser Vino Tinto, al menos un 30% debe ser Vino Rosado, y al menos un 20% debe ser Vino Blanco.

f) La empresa ha decidido hacer zumo 100% natural, para ello se utilizará el agua que resulta de la fermentación y nada de agua externa.

g) Con la capacidad propia anterior, se podrían alquilar unas cubas de fermentación con un coste fijo de 100€ y un coste variable por hectolitro de 10€. ¿Cómo incorporar esta condición?

h) ¿Cómo incorporaría la condición de que o bien se compra uva de tipo 1 o se compra uva de tipo 2, pero no las dos simultáneamente?

## *15.8 Plan de Producción de Zapatillas*

Una empresa fabricante de zapatillas deportivas estima la siguiente demanda (en centenares de pares) para los próximos 6 meses.

| | Mes 1 | Mes 2 | Mes 3 | Mes 4 | Mes 5 | Mes 6 |
|---|---|---|---|---|---|---|
| Cantidad | 30 | 40 | 55 | 30 | 60 | 35 |

Los costes de producción de cada par se evalúan en 7€ si se producen en horas normales, y 9€ si se produce en horas extras. El número de horas normales disponibles por día es de 8 horas. El número de días laborables por mes es de 20. Cada mes se puede trabajar un máximo de 40 horas. Durante cada hora de producción es posible fabricar 25 pares de zapatillas. El coste de almacenar un par de zapatillas de un mes para otro es de 0.1€ por par. Se dispone de 200 pares en stock en estos momentos.

a) Establecer el modelo que permite definir el plan de trabajo para los próximos meses a mínimo coste.

b) Suponga que tiene un compromiso con los trabajadores de utilizar cada dos meses al menos un 25% de la capacidad en horas extra actual.

c) ¿Cómo limitar la capacidad del almacén?

d) Suponga que el objetivo sea minimizar el máximo stock entre periodos.

e) Suponga que el objetivo es minimizar las diferencias de producción entre un mes y el siguiente: para cualquier mes.

f) ¿Cómo modelaría que se pretende minimizar la máxima desviación entre la producción y la demanda?

g) ¿Cómo haría periodos de duración variable? Por ejemplo la consideración de meses de diferente cantidad de días.

h) ¿Cómo incorporaría diferentes tipos de zapatillas en el modelo, para poder hacer un plan conjunto?

i) Suponiendo e) ¿Cómo incorporaría un precio fijo a fabricar de un tipo o de otro en cada periodo de tiempo? *Básicamente es como un coste de cambio de partida, si se fabrica el tipo A, se paga una penalización dependiente de A.*

j) ¿Cómo limitaría los tipos de producto a fabricar en cada periodo de tiempo?

k) Sobre el modelo original ¿podría incorporar una lista de materiales del producto a fabricar, para establecer sus planes de compra? Suponga un plazo de entrega de un mes para cada componente.

l) Plantear el dual del problema original.

## *15.9 Problema de Distribución*

Una empresa de fabricación y distribución distribuye a 5 zonas de clientes, a través de 3 centros de distribución que se aprovisionan de 2 destilerías, propiedad también de la misma empresa.

Se conocen los costes de servir a cada zona de clientes desde cada destilería a través de cada centro de distribución. Cada zona de clientes tiene una demanda *D* que debe ser abastecida. Cada destilería tiene una capacidad *C* de producción limitada. Cada centro de distribución tiene una capacidad limitada tanto superior *S* como inferiormente *I* , y unos costes fijos *F* anuales y otros relativos al volumen *V* que se mueve.

La empresa se plantea la política de mínimo coste para el nuevo año, lo que podría implicar cerrar algunos de los centros de distribución.

a) ¿Cómo modelaría el problema?

b) ¿Cómo incorporaría la consideración de diferentes productos en el mismo sistema logístico?

c) ¿Cómo definiría un modelo que pretendiera minimizar la diferencia de uso de capacidad entre las dos destilerías, siempre que no se sobrepasaran los costes totales óptimos más que en un 20%?

d) ¿Cómo incorporaría la posibilidad de servir directamente a las zonas de clientes desde las destilerías?

e) De los 3 centros de distribución del modelo original sólo pueden quedar 2. Modele la situación.

f) Se plantea hacer una inversión en el primer centro de distribución, para ampliar su capacidad en un 40%. ¿Cómo evaluaría las ventajas de invertir o no invertir?

g) Se va a hacer una inversión en algún centro de distribución, para ampliar su capacidad en un 40%. ¿Cómo evaluaría en cuál de ellos compensa invertir?

h) Se plantea hacer una inversión en el primer centro de distribución, para ampliar su capacidad en un 40%. ¿Cómo evaluaría las ventajas de invertir o no invertir?

i) Se plantea abrir un nuevo centro de distribución. Con unos costes de construcción, amortizables en 5 años, y unos costes de funcionamiento Fijos y Variables. ¿Cómo plantear el problema para saber si será rentable y a quien afectará el cambio?

j) Sobre el problema original se pretende maximizar el ratio de costes variables sobre fijos. Modele el problema.

k) Sobre el problema original se pretende que la capacidad de la destilería primera pueda duplicar, o triplicar su capacidad a diferentes costes. Modele el problema.

l) La capacidad de la destilería 2 puede ser superada con un coste adicional por unidad vendida.

m) Los costes variables de trasiego en cada centro de distribución, crecen proporcionalmente a la raíz cuadrada del volumen movido. Modele la situación.

## *15.10 PKJu Electricidad*

Trabaja usted en una empresa cuya actividad básica es la compra-venta de material electrónico. La empresa ha decidido implantar un sistema de gestión de stocks informatizado, que decida cuánto y cuándo comprar.

Ha quedado usted encargado de ello, entre otros motivos, por no asistir a la reunión donde se trató el tema. Sus compañeros insisten en que el *marrón* no le cayó por no estar presente, sino porque usted es el más capacitado para sacarlo adelante.

Tras analizar los productos con los que trabaja (demandas variables, precios variables, costes de almacenamiento variables, etcétera) decide que no puede aplicar métodos de punto de pedido o de aprovisionamiento periódico.

Para desarrollar un sistema informático, usted pretende plantear un modelo matemático y posteriormente resolver con herramientas de las que existen en la *red.*

La empresa compra y vende un solo producto[6] de un proveedor taiwanés. El tipo de entrega es en *bodegas del cliente,* esto es, usted paga un precio de compra (**PC**) por el producto (que es variable para los distintos periodos de tiempo) y el proveedor se hace cargo de todo el transporte, aduanas y demás gastos asociados.

El plazo de entrega desde que se lanza el pedido es de 4 semanas. Y al cliente se le paga 4 semanas después de haber recibido el producto.

Los costes de almacenamiento físico (**H**) son conocidos aunque varían según el periodo. El dato viene en unidades monetarias por unidad de producto almacenado y por periodo de previsión.

La demanda (**D**) de nuestro producto varía con los periodos, aunque su variabilidad es conocida con un margen de error pequeño. El precio de venta es conocido y variable por periodo (**PV**). La empresa suele cobrar 3 semanas después de haber producido la venta.

El saldo negativo del flujo de caja (diferencia entre cobros y pagos) tiene un coste financiero (**CF**), del mismo modo que el saldo positivo lleva asociado un coste de oportunidad (**CO**). Además no es posible sobrepasar el saldo negativo en más de una cierta cantidad límite (**LN**). La empresa, como cualquier empresa seria, tiene una previsión de pagos (**PP**), que es variable según los periodos. Esta previsión incluye los gastos como nóminas, beneficios, energía, alquileres pero excluye los gastos de adquisición de materiales (porque de esto se encarga su programa informático).

[6] Evidentemente esto es una simplificación para empezar.

Como es evidente, el objetivo de la empresa es ganar la máxima cantidad de dinero posible[7]. Los periodos de previsión que usted considera son semanas. Y el horizonte de planificación es de 26 semanas.

a) Plantee el modelo matemático. Si requiere incorporar variables y/o parámetros no dude en hacerlo. Se considerará que indique y explique los índices, parámetros y variables que va a utilizar. Se considerará que explique cada restricción. Se agradecerá que utilice mayúsculas para los parámetros y minúsculas para las variables, asimismo será agradecido el uso de los $w_i$: flujo de caja negativo en el periodo t

b) ¿Cómo se modifica el modelo si hubiera más de un producto que se recibe del mismo proveedor? Construya el modelo.

c) Suponga que si en un periodo se produce compra de alguno de los productos hay que pagar un coste fijo (**SU**) por transporte. Modele la situación.

d) Suponga que puede diferir demanda de un periodo a otro con un coste suplementario ($K_{i,t}$) por unidad diferida por periodo. El ingreso que se producirá será acorde al precio de venta en la semana en que el cliente quería adquirir el producto.

e) Suponga que hay descuentos por cantidad para cada producto, de tal modo que los precios de compra por unidad son los siguientes:

$$PC_i = \begin{cases} A_i & \text{si la cantidad a comprar es menor que } 100 \\ B_i & \text{si la cantidad a comprar esta en } [100,500[ \\ C_i & \text{si la cantidad a comprar esta en } [500,1000[ \\ D_i & \text{si la cantidad a comprar es mayor que } 1000 \end{cases}$$

Modele la situación, respetando la linealidad del modelo.

**f)** Un gran distribuidor le ofrece un contrato de compra garantizada para el próximo año de (**$CG_t$**) unidades de uno de los productos en cada uno de los periodos. Sus condiciones son: f1) que el precio por el que el distribuidor compra ha de ser un 25% inferior a nuestro precio de venta habitual y

f2) los plazos de pago son de 9 semanas desde que el distribuidor adquiere el producto.

Sus estimaciones indican que de aceptar el trato este distribuidor se hará con el 10% de su demanda habitual estimada, que le comprará a él en lugar de a usted. Si no aceptase la oferta sabe que perderá un 5% de las ventas. Modele la situación.

## *15.11 Equilibrado de Líneas*

Se tiene un conjunto de tareas, cada una con una duración y un conjunto de precedencias.

[7] Hay que recordar que en contabilidad se aplica el concepto del devengo, que distingue el cobro del ingreso, y los gastos de los pagos. Para saber cuánto dinero gana una empresa nos fijamos en lo que ingresa y lo que gasta. Para conocer el flujo de caja nos fijamos en lo que cobra y lo que paga.

a) Dado un tiempo de ciclo. Establecer una asignación que utilice el mínimo número de estaciones.
b) ¿Cómo conseguir además el mínimo desequilibrio de carga entre estaciones?
c) Dado el número de estaciones, conseguir la asignación de mínimo tiempo de ciclo.
d) ¿Cómo obligar a que dos tareas determinadas estén separadas?
e) ¿Cómo obligar a que dos tareas determinadas estén juntas?

## *15.12 Jorge y Nuria*

Jorge y Nuria se quieren casar. Ella de blanco por la alegría y él de negro. Entre otros quebraderos de cabeza que les harán perder unas cuantas horas de aquí al día de la boda, no es el menor la tarea de asignar los invitados a las mesas. Inicialmente Jorge y Nuria querían una boda íntima, como todos, aunque ahora ya suman alrededor de 250 invitados.

Dicho reparto de invitados es una de las actividades más tediosas de las que se encontrarán: El tío tal no se puede sentar con su concuñado porque se llevan a muerte, dos invitados excusan en el último momento su asistencia...

Tienen un amigo, usted, gran experto en diseño de modelos e implementaciones informáticas, que les va a echar una mano, construyendo una aplicación que permitirá hacer y rehacer, modificar, imprimir las listas, etcétera.

Para ello considera que uno de los conjuntos de datos (en el modelo debe constituir uno de los índices) a considerar son las agrupaciones mínimas (individuos, matrimonios, parejas, tríos y otras agrupaciones de invitados). Estas agrupaciones mínimas son grupitos de invitados que no se pueden separar, por razones más o menos obvias.

Otro de los conjuntos de datos a considerar son las mesas. Cada una de ellas tiene una capacidad diferente (entre 8 y 12 comensales cada una).

a) Plantee un modelo donde se pretenda minimizar el número de mesas a utilizar.
b) Plantee una restricción que evite que dos cuñados que se llevan mal se sienten en la misma mesa.
c) Suponga que el número de mesas máximo viene fijado por la capacidad del salón. Suponga que entre cada agrupación mínima de invitados existe una relación de afinidad (que los novios valorarán numéricamente). Plantee un modelo que persiga maximizar la suma de afinidades en las mesas. Es decir, la afinidad entre dos agrupaciones mínimas se suma si pertenecen a la misma mesa. *Será interesante en este apartado incorporar una variable binaria que indique si cada par de agrupaciones mínimas está en una misma mesa.*

En el caso anterior ¿cómo penalizaría que una agrupación mínima se quedara *aislada* en una mesa sin conocidos?

## *15.13 Operación Brisca*

Usted trabaja en una multinacional del sector audiovisual y últimamente le están pidiendo que haga recopilatorios de canciones de "cantantes de ayer y siempre". Estos recopilatorios se graban en CD's para vender a través de Teletiendas. Básicamente el problema consiste en definir un conjunto de canciones que quepan en el CD, utilizando al máximo el espacio disponible y que las diferentes épocas del autor estén reflejadas. Últimamente está un poco harto del trabajo, en el que su criterio artístico (su famoso "oído musical") no sale demasiado bien parado. Está pensando en hacer un pequeño *software* basado en programación lineal que haga automáticamente las recopilaciones.

Para ello parte de los siguientes índices, datos y variables:

| Índices | Datos | Variables |
|---|---|---|
| *i*: Recorre las épocas del autor<br>*j*: Recorre las canciones de cada época | *C*: Capacidad del soporte<br>$d_{i,j}$: Duración de la canción j de la época i | $X_{i,j}$ : Variable binaria que vale 1 si la canción j de la época i se incorpora en el soporte, 0 en caso contrario |

a) Defina la función objetivo.

b) Defina la restricción de que las canciones seleccionadas deben caber en el soporte.

c) Si fijamos un número mínimo de canciones de cada época ¿cómo incorporaría esta restricción?

d) ¿Cómo minimizaría la diferencia entre el máximo y el mínimo número de canciones de cada época que ha puesto?

e) *Desde c)* Suponga que le piden que diseñe, con las mismas canciones, además, una cinta de audio para la venta en gasolineras (con dos caras de igual capacidad $C_a$). Diseñe el modelo entero incorporando lo que considere conveniente.

f) *Desde c)* Determinadas canciones no pueden entrar simultáneamente en el mismo CD por razones de copyright. Sean éstas la canción 3 de la época 1 y la canción 8 de la época 3.

g) *Desde c)* Por razones sentimentales le indican que la canción 4 de la época 2 y la canción 3 de la época 5 entran las dos o no entra ninguna.

h) *Desde c)* Suponga que pretende dotar de una cierta calidad al sistema. Para ello puntúa las diferentes canciones y pretende que la calidad del CD sea lo máxima posible. Modele la nueva función objetivo.

i) *Desde h)* La productora le comenta que lo de la calidad está muy bien, pero que las buenas canciones valen más dinero. Plantee el objetivo lineal (así como las restricciones asociadas) que permiten maximizar el ratio calidad/precio.

## *15.14 Carga de Aviones*

Es usted el responsable de carga de una empresa de transporte de mercancías por avión. Está intentando cargar un BAING 717 con una carga de 4 productos distintos. Cada uno de ellos con las siguientes características:

| Carga | Peso (Tm) | Volumen ($m^3$/Tm) | Beneficio (€/Tm) |
|---|---|---|---|
| C1 | 18 | 500 | 300 |
| C2 | 20 | 600 | 450 |
| C3 | 10 | 550 | 350 |
| C4 | 16 | 400 | 275 |

El proceso de asignación de cargas es como sigue. El avión se divide en 3 compartimentos (Delantero, Central y Trasero) con diferentes capacidades en peso y en volumen cada uno de ellos. Dichas capacidades no se pueden sobrepasar

| | Capacidad en Peso (Tm) | Capacidad en Volumen ($m^3$) |
|---|---|---|
| Delantero | 12 | 6000 |
| Central | 16 | 9000 |
| Trasero | 10 | 5000 |

Además y a efectos de estabilidad en vuelo del avión, es necesario que el porcentaje de peso ocupado sobre el total sea el mismo en cada compartimiento.

(1 punto) a) Defina la función objetivo.

(1 punto) b) Defina las restricciones que considere necesarias para establecer la limitación de capacidad en peso y volumétrica.

(1 punto) c) Defina las restricciones que considere necesarias para establecer la limitación de demanda.

(1 punto) d) Defina las restricciones que considere necesarias para establecer las consideraciones de equilibrio de la carga.

(1 punto) e) ¿Cuál sería el beneficio de aumentar un poco aún a costa de seguridad (ya sabe los beneficios son los beneficios) la carga en peso de la parte delantera?

(1 punto) f) ¿Cuál es el precio que debiera pagar el propietario de la carga de tipo 1 para que consideráramos su producto como posible mercancía en nuestro avión, sin pérdida de beneficios?

(1 punto) g) Hay algunos elementos prescindibles en la carga muerta del avión que ocupan volumen aunque no pesan nada, (paracaídas y demás). Su jefe le pregunta por el beneficio que supondría por unidad de volumen, deshacerse de ellos.

(1 punto) h) Un cliente le indica que si transporta carga de tipo 2 debe también transportar al menos un 25% de la carga de tipo 1. Modele la situación.

(1 punto) i) Defina el modelo que incorpore todas las restricciones citadas y además el objetivo sea maximizar el ratio beneficio total/volumen ocupado.

(1 punto) j) ¿Cómo permitiría un desequilibrio de un 10% en el ratio de peso entre las distintas bodegas del avión?

# 16 ALGUNOS EJERCICIOS DE OPTIMIZACIÓN COMBINATORIA

## 16.1 Problema del FlowShop de 3 máquinas

### 16.1.1 Descripción del Problema

Sea un taller de flujo compuesto por *M* máquinas donde se deben secuenciar *N* productos, que para su elaboración pasarán de manera consecutiva por cada una de las *M* máquinas, con unos tiempos de ejecución de cada producto *i* en cada máquina de *p(i,j)* respectivamente.

Se trata de definir la secuencia que minimice el tiempo de flujo máximo (*Cmax*) que se define como el lapso de tiempo mínimo que transcurrirá desde que el primer producto se empieza a producir en la primera máquina hasta que el último se acaba de producir en la última máquina.


### 16.1.2 Definición de la estructura de la solución

Una solución es la permutación que indica el orden en que se deben ejecutar los trabajos.

Una solución se puede representar por un vector *v[i]* que define de modo ordenado la secuencia de trabajo.

### 16.1.3 Definición del modo de evaluar la solución

Dada una solución *v(i)*, se trataría de calcular cuando acaba de producirse el producto en posición *n*, es decir *v(n).*

Ejemplo X

```
Maq[1]:=0;Maq[2]:=0;Maq[3]:=0;
for i=1 to n do begin
    Maq[1]:=Maq[1]+p[v[i],1];
    Maq[2]:=Max(Maq[1],Maq[2])+p[v[i],2];
    Maq[3]:=Max(Maq[2],Maq[3])+p[v[i],3];
end;
Cmax=Maq[3];
```

### 16.1.4 Un procedimiento de generación aleatoria de soluciones

Ejemplo X

```
Sea v un vector con los n elementos a permutar
i:=0;
While i<n-1 do begin
    j:=random(1...n-i)
    intercambiar(v[i+1];v[i+j])
```

```
    i:=i+1
end; // del while
```

### 16.1.5 Un procedimiento enumerativo de resolución

Un procedimiento enumerativo basado en ramificación y poda requiere definir la función de evaluación de cada nodo, definir el modo en que se va a seleccionar el nodo a explotar, definir el modo de explosión, definir el primero de los nodos y definir el modo de eliminar los nodos.

```
generarElNodoInicial;
generarUnaSoluciónInicial;
Mientras haya nodos abiertos;
Elegir un nodo (por ejemplo según menor f(n))
Explotar el nodo (añadir un producto de los no secuenciados a la secuencia).
Evaluar f(n) para los nuevos nodos.
Eliminar los nodos para los cuales el valor de f(n) es mayor que la mejor solución ya obtenida.
Si el número de elementos a añadir es 0, evaluar el nodo y comparar contra la mejor solución
obtenida. Guardar en su caso.
```

- *Definir f(n)*
  - *f(n)=g(n)+h(n)*
  - *g(n)=el momento en el que acaba el último producto de la secuencia ya generada en la última máquina*
  - *h(n)=la suma de las operaciones en la última máquina de los productos no secuenciados.*
- *Definir el modo de selección del nodo a explotar*
  - *Por ejemplo el de menor f(n)*
- *Definir el modo de explotar nodos*
  - *Añadir un producto aún no secuenciado a la secuencia.*
- *El primer nodo es la secuencia vacía.*

### 16.1.6 Un procedimiento heurístico

Se puede establecer un procedimiento heurístico por la vía de generar una lista ordenada según un determinado criterio y secuenciar según dicho criterio.

```
Sea v un vector con los n elementos a permutar
Calcular para cada producto prod[i]:=p[i,1]+p[i,2]+p[i,3];
Ordenar los productos según prod crecientes
Asignar a v la anterior ordenación.

Sea v un vector con los n elementos a permutar
Calcular para cada producto prod(i)=sum{j=1…3} (2j-1)*p(i,j)
Ordenar los productos según prod decrecientes
Asignar a v la anterior ordenación.
```

### 16.1.7 Un procedimiento de mejora local

Definir un procedimiento de mejora local exige: Seleccionar una definición de vecindario, Seleccionar un proceso de búsqueda local y seleccionar el modo de generación de una solución inicial.

El vecindario seleccionado es el de intercambiar dos elementos de la permutación.

El proceso de mejora local seleccionado es Mejora Iterativa Simple.

La solución inicial se genera aleatoriamente.

Ejemplo X

```
generarSolucionAleatoria;
coste:=evaluarSolución(v);
i:=1;
while (i<=n-1) do begin
    j:=i;
    HayMejora:=false;
    while ((j<=n) and not(HayMejora)) do begin
            j:=j+1;
        intercambiar(v[i],v[j]);
        if evaluarSolucion(v)<=coste then HayMejora:=true else
     deshacerIntercambio(v[i],v[j]);
    end;
    if j=n then i:=i+1;
    if HayMejora then i:=1;
end; // del while
```

### 16.1.8 Un algoritmo genético

La estructura de un algoritmo genético es:

```
generarPoblacionInicial;
while not(hayQueParar) do begin
    seleccionarPadres;
    construirNuevaSolucion;
    provocarMutaciones;
    actualizarPoblación;
end;
```

Por tanto para definir el procedimiento hace falta:

- Definir un tamaño de población inicial: 20.
- Definir un modo de generación de la población inicial: aleatoriamente.
- Definir un modo de parar: al cabo de 1000 iteraciones.
- Se selecciona un modo de selección de padres en función de una ruleta donde el valor de *fitness* sea la inversa de la valoración de la solución.
- Se define un procedimiento de cruce (por ejemplo OX con dos puntos de cruce).
- Se actualiza la población: Si una solución obtenida del cruce es mejor que la peor de las soluciones la sustituye. Si no es mejor se calcula una mutación mediante un intercambio de dos elementos y se incorpora a la población eliminando una solución cualquiera elegida al azar.

## *16.2 Problema del viajante de comercio*

### 16.2.1 Descripción del Problema

Sea un conjunto de puntos de un plano a una cierta distancia unos de otros.

Se trata de unir todos los puntos mediante un circuito que sólo visite cada nodo en una ocasión minimizando el recorrido.

### 16.2.2 Definición de la estructura de la solución

La solución se puede representar mediante una permutación de los puntos a visitar.

### 16.2.3 Definición del modo de evaluar la solución

```
coste:=0
For i:=1 to n-1 do begin
    coste=coste+distancia[v[i],v[i+1]];
End
coste=coste+distancia[v[n],v[1]];
```

### 16.2.4 Un procedimiento de generación aleatoria de soluciones

```
Sea v un vector con los n elementos a permutar
i:=0;
While i<n-1 do begin
    j:=random(1...n-i)
    intercambiar(v[i+1];v[i+j])
    i:=i+1
end; // del while
```

### 16.2.5 Un procedimiento enumerativo de resolución

Un procedimiento enumerativo basado en ramificación y poda requiere definir la función de evaluación de cada nodo, definir el modo en que se va a seleccionar el nodo a explotar, definir el modo de explosión, definir el primero de los nodos y definir el modo de eliminar los nodos.

```
generarElNodoInicial;
generarUnaSoluciónInicial;
Mientras haya nodos abiertos;
Elegir un nodo
Explotar el nodo.
Evaluar f(n) para los nuevos nodos.
Eliminar los nodos para los cuales el valor de f(n) es mayor que la mejor solución ya
obtenida.
Si el número de elementos que quedan por añadir es 0, evaluar el nodo y comparar
contra la mejor solución obtenida. Guardar en su caso.
```

- *Definir f(n)*
  - *f(n)=g(n)+h(n)*
  - *g(n)= es la suma de las distancias entre los puntos ya asignados*
  - *h(n)=el número de nodos por asignar multiplicado por el menor valor de la matriz distancia más el menor valor en la columna del primer elemento de la secuencia.*
- *Definir el modo de selección del nodo a explotar*
  - *Por ejemplo el de mayor profundidad(mayor número de ciudades incorporadas) en caso de empate utilizar el menor valor de f(n)*
- *Definir el modo de explotar nodos*
  - *Añadir todos los puntos aún no visitado a la secuencia.*
- *El primer nodo contiene un punto cualquier al azar al azar.*

### 16.2.6 Un procedimiento heurístico

Un procedimiento heurístico de construcción sería el del vecino más próximo.

Sea v un vector con todos los nodos a visitar

```
i:=1
while i<=n-1 do begin
    seleccion=i+1;
    for j:=i+2 to n do begin
        if distancia[v[i],v[j]]< distancia[v[i],v[seleccion]] then seleccion:=j;
    end; // del for
    intercambiar[v[i+1],v[seleccion]];
end; //del while
```

Este procedimiento podría ser mejorado si se repitiera N veces y en cada ocasión se hiciera empezar el ciclo por un nodo diferente y se guardara el mejor resultado. Además se podría comenzar el tour por el último nodo, aunque este procedimiento sólo produciría mejoras en caso de matrices asimétricas.

### 16.2.7 Un procedimiento de mejora local

Definir un procedimiento de mejora local exige: Seleccionar una definición de vecindario, Seleccionar un proceso de búsqueda local y seleccionar el modo de generación de una solución inicial.

El vecindario seleccionado es el de provocar un 2-opt (o mutación inversa).

El proceso de mejora local seleccionado es Mejora Iterativa Simple.

La solución inicial se genera mediante la Heurística Vecino más Cercano.

```
generarSolucionVecinoMasCercano;
coste:=evaluarSolución(v);
i:=1;
while (i<=n-1) do begin
    j:=i;
    HayMejora:=false;
    while ((j<=n) and not(HayMejora)) do begin
        j:=j+1;
        mutaciónInversa(i,j);
        if evaluarSolucion(v)<=coste then HayMejora:=true else
     mutaciónInversa(j,i);
    end;
    if j=n then i:=i+1;
    if HayMejora then i:=1;
end; // del while
```

### 16.2.8 Un algoritmo genético

La estructura de un algoritmo genético es:

```
generarPoblacionInicial;
while not(hayQueParar) do begin
    seleccionarPadres;
    construirNuevaSolucion;
    provocarMutaciones;
    actualizarPoblación;
end;
```

Definir un tamaño de población inicial: 20.

Definir un modo de generación de la población inicial: aleatoriamente.

Definir un modo de parar: al cabo de 1000 iteraciones.

Se selecciona un modo de selección de padres en función de una ruleta donde: *fitness*(v)=1/Evaluación(v).

Se define un procedimiento de cruce (por ejemplo OX).

Se define un procedimiento de actualización: De cada 100 iteraciones se guardan las 18 mejores, dos soluciones se calculan aleatoriamente y se incorporan a la población.

Cada 20 iteraciones se elige al azar un elemento de la población y se le hace una mutación inversa de modo aleatorio.

```
generarPoblacionInicial;
while not(hayQueParar) do begin
    seleccionarPadres;
    construirNuevaSolucion;
    provocarMutaciones;
    actualizarPoblación;
end;
```

## *16.3 Problema de Secuenciación JIT*

### 16.3.1 Descripción del Problema

Sea una línea de montaje donde se ensamblan 3 modelos de un determinado producto. De cada uno de los modelos hay que fabricar una cantidad *U[i]* determinada. Se pretende, para garantizar la regularidad, que es básica en los sistemas *Just-In-Time*, que la producción salga de modo tan equilibrado como sea posible.

### 16.3.2 Definición de la estructura de la solución

Una solución es una concatenación de los diferentes tipos de productos, de tal manera que la cantidad de cada tipo es al final igual a la demandada *U[i]*.

### 16.3.3 Definición del modo de evaluar la solución

El criterio más habitual para medir la calidad de una solución es intentando evaluar la distancia que separa la producción acumulada en un determinado instante de la secuencia, de la que idealmente habría que haber fabricado.

$$\sum_i \left(Y_{ik} - r_i\right)^2$$

Donde $Y_{i,k}$ es la producción acumulada hasta $k$ del producto de tipo $i$.

$r_i$ es el ratio de producción de $i$, $r_i=U_i / T$

*T* es el número total de unidades a producir, es decir la suma de $U_i$ para todo *i*.

### 16.3.4 Un procedimiento de generación aleatoria de soluciones

```
k:=0;
for i:=1 to P do begin
    for h:=1 to U[i] do begin
            k:=k+1;
            v[k]:=I;
     end;
end;
T:=k;
h:=0;
While h<T-1 do begin
    j:=random(1...T-h)
    intercambiar(v[h+1];v[h+j])
    h:=h+1
end; // del while
```

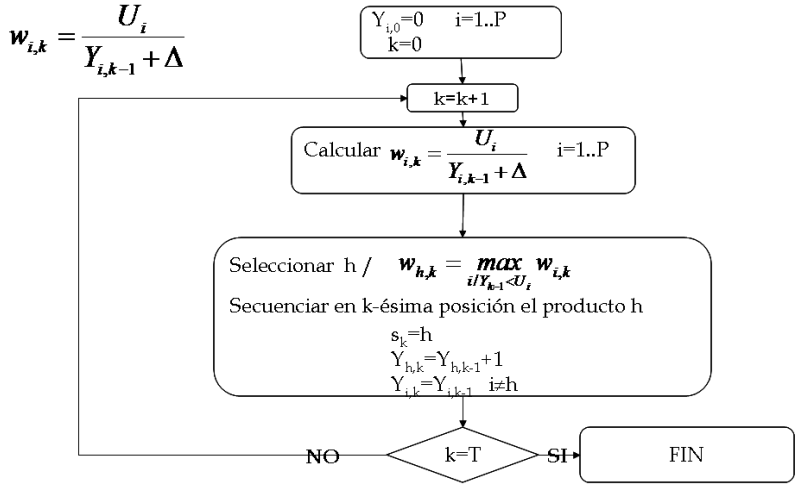
### 16.3.5 Un procedimiento enumerativo de resolución

Un procedimiento enumerativo basado en ramificación y poda requiere definir la función de evaluación de cada nodo, definir el modo en que se va a seleccionar el nodo a explotar, definir el modo de explosión, definir el primero de los nodos y definir el modo de eliminar los nodos.

```
generarElNodoInicial;
generarUnaSoluciónInicial;
Mientras haya nodos abiertos;
    Elegir un nodo
    Explotar el nodo.
    Evaluar f(n) para los nuevos nodos.
    Eliminar los nodos.
    Si el número de elementos que quedan por añadir es 0, evaluar el nodo y
    comparar contra la mejor solución obtenida. Guardar en su caso.
```

- *Definir f(n)*
  - *f(n)=g(n)+h(n)*
  - *g(n)= es la valoración de la función hasta el punto de creación.*
  - *h(n)=0.*
- *Definir el modo de selección del nodo a explotar.*
  - *Por ejemplo el de menor profundidad (menor número de unidades ya incorporados) en caso de empate utilizar el menor valor de f(n).*
- *Definir el modo de explotar nodos.*
  - *Añadir un producto (todos los productos) de los que no se han fabricado toda la secuencia.*
- *Eliminar aquellos nodos donde la cantidad de productos ya secuenciados de cada tipo sea idéntico a otro con mejor valoración.*
- *El primer nodo está vacío.*

### 16.3.6 Un procedimiento heurístico




La modificación del valor de Δ permitirá obtener diferentes secuencias con el mismo procedimiento.

### 16.3.7 Un procedimiento de mejora local

Definir un procedimiento de mejora local exige: Seleccionar una definición de vecindario, Seleccionar un proceso de búsqueda local y seleccionar el modo de generación de una solución inicial.

El vecindario seleccionado es el de provocar una inserción de un elemento elegido al azar en una posición elegida al azar.

El proceso de mejora local seleccionado es Descenso Rápido.

La solución inicial se genera aleatoriamente.

```
generarSolucionVecinoMasCercano;
coste:=evaluarSolución(v);
i:=1;
while (i<=n-1) do begin
    j:=i;
    HayMejora:=false;
    while ((j<=n) and not(HayMejora)) do begin
            j:=j+1;
        mutaciónInversa(i,j);
        if evaluarSolucion(v)<=coste then HayMejora:=true else
     mutaciónInversa(j,i);
    end;
    if j=n then i:=i+1;
    if HayMejora then i:=1;
end; // del while
```

### 16.3.8 Un algoritmo genético

La estructura de un algoritmo genético es:

```
generarPoblacionInicial;
while not(hayQueParar) do begin
    eleccionarPadres;
    construirNuevaSolucion;
```

```
    provocarMutaciones;
    actualizarPoblación;
end;
```

Por tanto para definir el procedimiento hace falta:

Definir un tamaño de población inicial: 20.

Definir un modo de generación de la población inicial: aleatoriamente.

Definir un modo de parar: al cabo de 1000 iteraciones.

Se selecciona un modo de selección de padres en función de una ruleta donde el valor de *fitness* sea la inversa de la valoración de la solución.

Se define un procedimiento de cruce: en este caso se propone un OX con un punto de cruce, donde la sección del Padre y la Madre se incorporan íntegramente y se repara (aquellos productos que se producen de más frente a los que se producen de menos) aleatoriamente quitando algún producto de los que hay de más para poner alguno de los que hay de menos (ambos seleccionados aleatoriamente).

Se actualiza la población: Si una solución obtenida del cruce es mejor que la peor de las soluciones la sustituye. Si no es mejor se calcula una mutación mediante un intercambio de dos elementos y se incorpora a la población eliminando una solución cualquiera elegida al azar.

## *16.4 Corte de Piezas rectangulares*

En una empresa de fabricación de piezas metálicas, tienen una sierra automática capaz de hacer cualquier tipo de corte rectangular.

La citada empresa compra planchas rectangulares de dimensiones DX y DY.

La explosión de la lista de materiales de la empresa a partir de los pedidos en firme genera un conjunto de N piezas rectangulares de dimensiones *(a,b)*.

Se trata de diseñar un programa de corte para cada plancha que minimice los restos de cada plancha.

a) Construya el Modelo que define el problema de corte.

*Indices*

$i, j$ : Recorre los cortes

*Parametros*

$DX, DY$ : Dimensiones X e Y de la Placa Origen

$a_i, b_i$ : Dimensiones x e y de cada corte i

*Variables*

$x_i, y_i$ : Posicion (x,y) de la esquina superior izquierda de la placa i

$r_i, s_i$ : Posicion (x,y) de la esquina inferior derecha de la placa i

$\alpha_{ij} \in \{0,1\}$ : =1 si un producto i esta mas a la izquierda que otro j (0 en caso contrario)

$\beta_{ij} \in \{0,1\}$ : =1 si un producto i esta mas a arriba que otro j (0 en caso contrario)

$\gamma_i \in \{0,1\}$ : =1 si la orientacion del producto es la natural (0 si es la perpendicular)

$t$ : maximo de todos los valores $r_i$

$w$ : maximo de todos los valores $s_i$

$$[Maximize] \quad (DX - t) \cdot DY + (DY - w) \cdot DX - w \cdot t$$

Sujeto a:

$$r_i = x_i + b_i \cdot \gamma_i + a_i (1 - \gamma_i) \qquad \forall i$$

$$s_i = y_i + a_i \cdot \gamma_i + b_i (1 - \gamma_i) \qquad \forall i$$

1

$$t \geq r_i \qquad \forall i$$

$$w \geq s_i \qquad \forall i$$

$$r_i \leq DX \qquad \forall i$$

$$s_i \leq DY \qquad \forall i$$

$$\alpha_{ij} = 1 \rightarrow (x_i + a_i \leq x_j) \qquad \forall i \neq j$$

$$\beta_{ij} = 1 \rightarrow (y_i + b_i \leq y_j) \qquad \forall i \neq j$$

$$\alpha_{ij} + \alpha_{ji} + \beta_{ij} + \beta_{ji} \geq 1 \qquad \forall i \neq j$$

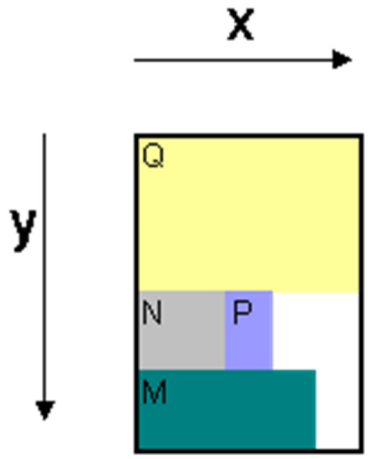
$$x_i, y_i, r_i, s_i \geq 0$$

2 Sin que sirva de precedente se incorpora un ejemplo numérico)

3 DX=5 DY=8

4 Pieza M: (4,2) ; Pieza N: (2,2) ; Pieza P: (2,1) ; Pieza Q: (4,5)

5 Una posible solución gráfica al problema sería




La solución anterior se puede definir del siguiente modo: (Q1,N0,P1,M0), que representa que primero se ubica la pieza Q en posición perpendicular a la definida, luego N en su posición natural en la esquina más arriba y más a la izquierda posible. Luego el Producto P en posición perpendicular, y el producto M en posición natural.

b) Sobre el resultado anterior. ¿Cuál es el resultado de ejecutar una inserción del 4 elemento en la segunda posición?

(Q1,M0,N0,P1)

c) Sobre el resultado anterior. ¿Cuál es la solución de ejecutar una 2-permutación del primer elemento con el tercero?

(N0,M0,Q1,P1)

d) ¿Cuál es la solución resultado de ejecutar un crossover (operador de los algoritmos genéticos) entre la solución (M0,N0,P1,Q1) y (M1,P1,Q0,N0) fijando las posiciones 2 y 3

(M1,N0,P1,Q0)

## *16.5 Quinielas*

Después de mucho pensar ha llegado a la elemental conclusión de que trabajando y ahorrando nadie se hace rico.

Así pues, se ha lanzado por el terreno legal más cómodo para hacer dinero sin dar ni golpe: las apuestas. Concretamente las Quinielas, porque le permitirán hacer uso de su conocimiento exhaustivo del mundo del fútbol (tantas horas invertidas en leer el Marca más pronto o más tarde iban a dar sus frutos).

Pero su manera de ganar dinero será científica, y así tantas horas invertidas en Métodos Cuantitativos también serán útiles.

Quiere hacer un modelo matemático que rellene una columna en la quiniela con un número limitado de apuestas (todavía no es rico), maximizando la probabilidad de acertar con unas limitaciones que irá incorporando.

Sea $\alpha_{i,1}$ la variable que indica que apuesta por la victoria del equipo de casa en el partido *i*.

Sea $\alpha_{i,X}$ la variable que indica que apuesta por el empate en el partido *i*.

Sea $\alpha_{i,2}$ la variable que indica que apuesta por la victoria del equipo foráneo en el partido *i*.

(1 punto) a) Modele una restricción (o juego de ellas) que asegure que apuesta en todos los partidos.

(1 punto) b) Modele una restricción (o juego de ellas) que mostrará que no quiere apostar por menos de 4 *unos* ni por más de 7 *unos.*

(1 punto) c) Modele que si le pone un 1 a un determinado partido A, le debe poner un 2 a un determinado partido B, aunque no necesariamente a la inversa.

(2 puntos) d) Modele una limitación superior de 5 dobles y 2 triples.

(1 punto) e) Establezca una restricción que limite el número total de apuestas a 2000.

(1 punto) f) Incorpore una función objetivo que maximice la probabilidad de acertar.

(2 puntos) g) Un análisis estadístico previo parece indicar que el ingreso esperado no crece linealmente con las apuestas jugadas sino en una curva en S con los siguientes puntos (en escala logarítmica). Establezca un objetivo lineal que pretenda maximizar el ratio beneficio esperado frente a coste de las apuestas.

| Log(número de apuestas) | Log(coste de las apuestas) | log(Beneficio Esperado) |
|---|---|---|
| 0 | -0,52 | 0,04 |
| 1 | 0,48 | 0,30 |
| 2 | 1,48 | 1,00 |
| 3 | 2,48 | 3,00 |
| 4 | 3,48 | 4,48 |

| | | |
|---|---|---|
| 5 | 4,48 | 5,30 |
| 6 | 5,48 | 5,78 |
| 6,68 | 6,16 | 5,85 |

(1 punto) h) Una determinada combinación se puede representar mediante una secuencia de 14 letras donde A representa un 1, B representa una X, C representa un 2, D representa 1X, E representa 12, F representa X2 y G representa 1X2. Sea una posible solución al problema planteado ADEABCEFFBEBFB. Y sea otra posible solución al mismo problema EDABBAEGAABCFF. ¿Qué combinación obtendría aplicando el operador genético de *Order Crossover* (OX) o Cruce, con el corte en las posiciones 5 y 9?

**Notas de apoyo**

*Como sabe la Quiniela es un juego donde se deben acertar el máximo número de resultados de 14 partidos de fútbol (por el momento prescindiremos del 15). El resultado puede ser 1, X o 2. Es decir gana el de casa, empatan o gana el de fuera.*

*Tras su estudio de la excelente prensa deportiva y el análisis de vídeos, enfermerías, árbitros y previsiones climatológicas, usted establece una probabilidad de que en cada uno de los partidos se produzca uno de los resultados ($P_{i,1}$ es la probabilidad de que en el partido i gane el equipo de casa, $P_{i,X}$ es la probabilidad de que en el partido i empaten y $P_{i,1}$ es la probabilidad de que en el partido i gane el equipo visitante, evidentemente $P_{i,1}+ P_{i,X}+ P_{i,2}=1$).*

*Tenga en cuenta para el punto f) que en cada partido la probabilidad de acertar es la suma de las probabilidades de los signos jugados. Y además que la probabilidad de acertar la quiniela es la multiplicación de probabilidades. Y además, si $X_i$ es la probabilidad de acertar el partido i*

$$[Maximize]\prod_i X_i \text{ es equivalente a } [Maximize]\sum_i \log(X_i)$$

*Además la función logaritmo se puede aproximar por este conjunto de puntos:*

| X | log(X) |
|---|---|
| 0,01 | -2 |
| 0,1 | -1 |
| 0,3 | -0,52 |
| 1 | 0 |

*El número de apuestas jugadas es la multiplicación para todos los partidos del número de columnas jugadas en cada partido. Así, si para un partido jugamos un doble entonces el número de apuestas se multiplica por 2, y si jugamos un triple se multiplica por 3. Si el partido se juega a simple el número de apuestas se multiplica por 1.*

*Recuerde que* log(A*B)=log (A)*log(B) y log(A/B)=log (A) – log (B)

*Log(1)=0 ; log(2)= 0,30 ; log(3)=0,48*

***Resolución***

a) Modele una restricción (o juego de ellas) que asegure que apuesta en todos los partidos.

$$\alpha_{i,1} + \alpha_{i,X} + \alpha_{i,2} \geq 1$$

b) Modele una restricción (o juego de ellas) que mostrará que no quiere apostar por menos de 4 *unos* ni por más de 7 *unos.*

$$\sum_i \alpha_{i,1} \leq 7$$
$$\sum_i \alpha_{i,1} \geq 4$$

c) Modele que si le pone un 1 a un determinado partido A, le debe poner un 2 a un determinado partido B, aunque no necesariamente a la inversa.

$$\alpha_{A,1} = 1 \quad \rightarrow \quad \alpha_{B,2} = 1$$
$$\alpha_{B,2} \geq \alpha_{A,1}$$

d) Modele una limitación superior de 5 dobles y 2 triples.

Definimos 3 variables binarias para cada partido ($\delta,\ \varepsilon,\ \phi$) que indicarán si es simple doble o triple.

$$y_i = \alpha_{i,1} + \alpha_{i,X} + \alpha_{i,2}$$
$$y_i = \delta_i + 2\varepsilon_i + 3\phi_i$$
$$1 = \delta_i + \varepsilon_i + \phi_i$$
$$\sum_i \phi_i \leq 2$$
$$\sum_i \varepsilon_i \leq 5$$

e) Establezca una restricción que limite el número total de apuestas a 2000.

Definimos 3 variables binarias para cada partido *($\delta$, $\varepsilon$, $\phi$)* que indicarán si es simple doble o triple.

$$y_i = \alpha_{i,1} + \alpha_{i,X} + \alpha_{i,2}$$
$$y_i = \delta_i + 2\varepsilon_i + 3\phi_i$$
$$1 = \delta_i + \varepsilon_i + \phi_i$$
$$numero\ de\ apuestas = 2^{numdobles} * 3^{numtriples} \leq 2000$$
$$\log\left(2\sum_i \varepsilon_i\right) + \log\left(3\sum_i \phi_i\right) \leq \log(2000)$$

f) Incorpore una función objetivo que maximice la probabilidad de acertar.

Sea $X_i$ la probabilidad de acertar un determinado partido *i*:

$$X_i = \sum_j P_{ij}\alpha_{ij}$$

Sea $\mathrm{V}_i$ una variable equivalente a $\log(X_i)$

$$[Maximize] \quad \prod_i X_i \equiv [Maximize] \quad \sum_i \log(X_i) \equiv [Maximize] \quad \sum_i \mathrm{V}_i$$

$$Sujeto\ a:$$

$$X_i = 0,01\lambda_{1,i} + 0,1\lambda_{2,i} + 0,3\lambda_{3,i} + \lambda_{4,i} \quad \forall i$$
$$V_i = -2\lambda_{1,i} - \lambda_{2,i} - 0,52\lambda_{3,i} \quad \forall i$$
$$\lambda_{1,i} + \lambda_{2,i} + \lambda_{3,i} + \lambda_{4,i} = 1 \quad \forall i$$

g) Un análisis estadístico previo parece indicar que el ingreso esperado no crece linealmente con las apuestas jugadas sino en una curva en S en escala logarítmica. Establezca un objetivo lineal que pretenda maximizar el ratio beneficio esperado frente al coste de las apuestas.

$$[Maximize] \quad \left(\frac{Beneficio}{Coste}\right) \equiv [Maximize] \quad \log\left(\frac{Beneficio}{Coste}\right) \equiv [Maximize] \ \log(\mathrm{Beneficio}) \text{-} \log(\mathrm{Coste})$$

Sea *A* el logaritmo del número de apuestas, sea *C* el logaritmo del coste de las apuestas y sea B el logaritmo del Beneficio esperado

$$[Maximize](B - C)$$

$$Sujeto\ a:$$

$$B = 0,04\varphi_1 + 0,30\varphi_2 + \varphi_3 + 3\varphi_4 + 4,48\varphi_5 + 5,30\varphi_6 + 5,78\varphi_7 + 5,85\varphi_8$$
$$C = -0,52\varphi_1 + 0,48\varphi_2 + 1,48\varphi_3 + 2,48\varphi_4 + 3,48\varphi_5 + 4,48\varphi_6 + 5,48\varphi_7 + 6,16\varphi_8$$
$$A = \log\left(2\sum_i \varepsilon_i\right) + \log\left(3\sum_i \phi_i\right)$$
$$y_i = \alpha_{i,1} + \alpha_{i,X} + \alpha_{i,2} \quad \forall i$$
$$y_i = \delta_i + 2\varepsilon_i + 3\phi_i \quad \forall i$$
$$1 = \delta_i + \varepsilon_i + \phi_i \quad \forall i$$

h) Una determinada combinación se puede representar mediante una secuencia de 14 letras donde A representa un 1, B representa una X, C representa un 2, D representa 1X, E representa 12, F representa X2 y G representa 1X2. Sea una posible solución al problema planteado ADEABCEFFBEBFB. Y sea otra posible solución al mismo problema EDABBAEGAABCFF. ¿Qué combinación obtendría aplicando el operador genético de *Order Crossover* (OX) o Cruce, con el corte en las posiciones 5 y 9?

ADEA***BCEFF***BEBFB

***EDAB***BAEGA***ABCFF***

**EDABBCEFFABCFF**

## *16.6 SUDOKU*

Durante este corto verano del 2005 ha triunfado como pasatiempo el denominado SUDOKU. La mayor parte de los periódicos nacionales e internacionales llevaban diariamente uno (o más) SUDOKUs, que enganchaban (o del que se decían enganchados) cientos de bañistas en cualquiera de las playas de esta nuestra comunidad.

Pocos de estos bañistas sabían que el SUDOKU es un clásico problema matemático heredero del "cuadrado mágico" y del "cuadrado latino". Los cuadrados mágicos se remontan al 2800 AC en China, aunque posteriormente aparecieron también el Antiguo Egipto y otros lugares. Consiste en definir los valores de una matriz *NxN* de tal manera que ninguna de las celdas repita el valor y la suma de los valores en cada fila y en cada columna (así como los de las dos diagonales principales) valgan lo mismo. A ese valor se le denomina constante mágica.

Los cuadrados latinos sin embargo son más recientes, hubo que esperar 4600 años a que Euler los describiera. Los cuadrados latinos son matrices de *NxN* celdas en las que *N* elementos se deben asignar a cada celda, para que, en ninguna fila y en ninguna columna, se repita el mismo elemento dos veces (son especialmente útiles en el diseño de experimentos).

El Sudoku es un cuadrado latino de 9x9 donde se deben rellenar las celdas con los dígitos del 1 al 9 y no sólo se prohíben las repeticiones en filas y columnas, sino que además se prohíbe la repetición en el interior de 9 zonas disjuntas de tamaño 3x3. Además se asignan algunos valores en algunas celdas, y de este modo se pretende que haya una y sólo una solución al problema.

Por su configuración el Sudoku es también un problema de coloreado de grafos, donde cada celda es un nodo, que está unido con todos los nodos de su línea, de su columna y de su cuadrado.

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 5 | 3 | | | 7 | | | | |
| 6 | | | 1 | 9 | 5 | | | |
| | 9 | 8 | | | | | 6 | |
| 8 | | | | 6 | | | | 3 |
| 4 | | | 8 | | 3 | | | 1 |
| 7 | | | | 2 | | | | 6 |
| | 6 | | | | | 2 | 8 | |
| | | | 4 | 1 | 9 | | | 5 |
| | | | | 8 | | | 7 | 9 |

Se pide:

a) (3 puntos) Plantear un modelo matemático cuya solución sea una solución de un cuadrado mágico de 3x3 con constante mágica tan pequeña como sea posible.
b) (2 puntos) Plantear un modelo matemático para el problema de un cuadrado latino de 4x4. (Atención : Este es un problema sin objetivo)
c) (1 punto) El número de cuadrados latinos de orden 4 es muy elevado, nos interesa especialmente aquel que tiene los valores pequeños tan cerca de la celda superior izquierda como sea posible. Defina el objetivo del nuevo modelo.
d) (2 puntos) Plantear un modelo matemático cuya solución sea una solución del SUDOKU. Así ahorraremos mucho tiempo a los bañistas que podrán dedicarse al clásico autodefinido. (Nuevamente este modelo no tendrá objetivo, sólo restricciones).

Dado que no todos los bañistas disponen de CPLEX®, plantear un procedimiento de ramificación y corte que permita obtener una solución. El procedimiento debe ser lo más eficaz posible. Ramifica en cada nodo teniendo en cuenta los valores que puede asignar a cada celda, y elimina el nodo si al explotarlo no hay ninguna posibilidad.

e) (1 punto) Proponga un método para seleccionar el siguiente nodo a explotar.
f) (1 punto) Proponga un método para explotar un nodo.
g) (1 punto) ¿Qué información debiera tener en cuenta para que el descarte de opciones al abrir cada nuevo nodo sea lo más rápido posible? Proponga un modo de almacenar dicha información.
h) (1 punto) Desarrolle el árbol para las cuatro primeras celdas que vaya a generar.

**Nota:** *(Esta nota es útil a partir del apartado b)* Un modo de garantizar que todos los elementos son distintos, y pertenecientes a un conjunto limitado de opciones, es establecer un índice para cada una de las opciones. De este modo se puede definir la opción a introducir en cada celda asignando a una variable binaria, que identifique la celda y la opción que se le ha incorporado, con el valor 1 (y 0 en caso de que no se asigne dicho valor). Así se podría definir una variable binaria $\lambda_{ijk}$ que valiera 1 si en la celda de coordenadas *i* y *j* se asigna el valor *k*.

**Nota 2:** Si una restricción no sabe cómo representarla escríbala “en letra”, recuerde que el proceso y el orden del mismo es tan (o más) importante que el resultado.

## *16.7 Secuenciando en la Línea de Montaje*

Una línea de montaje de coches tiene en medio de su sistema un almacén formado por 4 líneas en paralelo cada una con capacidad para alrededor de 20 coches, que sirve para absorber las fluctuaciones entre la primera parte de la planta (denominada *Trim*) y la segunda parte de la misma (denominada chasis).

Los coches entran en el almacén y son distribuidos de acuerdo a un criterio que no es relevante en este momento. Los coches de una determinada línea no pueden adelantarse entre sí, es decir si se pretende sacar el tercer coche de una línea se debe secuenciar primero las unidades por delante de él.

Tras analizar la situación descubre que la calidad de la secuencia va relacionada con un concepto que se denomina "violación de restricciones". Por lo visto hay una serie de restricciones de diferentes niveles de importancia cada una. Una restricción se puede representar del siguiente modo: "No más de dos Fort Juerga con Lavafaros de cada 6 coches".

Cómo evalúa el cumplimiento de restricciones no es objeto del presente problema. Se admite que un evaluador actuaría del siguiente modo: cada coche <i> al ser incorporado a una secuencia S viola unos puntos que calculará una función G(S*<i>). Y alguien ya tiene desarrollado el evaluador.

Tras explorar diferentes alternativas decide que lo mejor es buscar en cada iteración del proceso (es decir cada vez que hay que sacar un coche) la mejor secuencia posible de 10 unidades y sacar el primer coche de dicha secuencia.

En una brillante idea se le ocurre que la representación de la solución más adecuada, no es la secuencia de unidades a extraer sino la secuencia de las líneas desde las que se van a extraer las unidades. Así una solución (4,2,1,1,2,1,3,4) sacaría primero el primer coche de la línea 4 (que evidentemente sabemos cuál es), luego el primer coche de la línea 2, luego el primero de la línea 1 seguido del segundo de la línea 1, posteriormente el segundo de la línea 2 seguido del tercero de la línea 1, etcétera.

Para sacar 10 coches tiene por tanto $4^{\wedge 10}$ combinaciones y por tanto desprecian (aunque no deberían) la posibilidad de hacer una enumeración completa de las soluciones.

Se le pide que:

a) Diseñe un algoritmo que genere una solución aleatoria.
b) Diseñe un algoritmo heurístico que genere una solución que podamos considerar correcta.
c) Diseñe un algoritmo de tipo Mejora Local.
d) Diseñe un algoritmo de tipo genético.
e) Modifique el algoritmo diseñado en b) teniendo en cuenta que en las líneas el número de unidades que puede haber puede ser muy bajo.

Cada apartado debe constar de dos partes. En la primera hay que describir el algoritmo, en la segunda hay que ejecutar el algoritmo poniendo un ejemplo.

Para diseñar el algoritmo hay que hacerlo de la manera más inteligible posible. Al ejecutar los algoritmos invéntese los datos que necesite. Si le hacen falta números aleatorios, aquí tiene una secuencia de números entre 0 y 99.

| | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 23 | 75 | 86 | 30 | 5 | 55 | 54 | 54 | 36 | 56 | 27 | 79 | 19 | 89 | 99 | 68 | 78 | 55 | 50 | 47 | 99 |
| 32 | 95 | 18 | 21 | 76 | 7 | 63 | 83 | 64 | 43 | 13 | 40 | 75 | 66 | 46 | 66 | 39 | 88 | 90 | 93 | 73 |
| 16 | 82 | 59 | 68 | 13 | 78 | 28 | 39 | 21 | 22 | 41 | 47 | 10 | 88 | 69 | 98 | 22 | 58 | 48 | 90 | 98 |
| 11 | 79 | 55 | 68 | 39 | 26 | 70 | 64 | 7 | 23 | 43 | 76 | 13 | 35 | 13 | 24 | 83 | 14 | 80 | 63 | 82 |
| 58 | 45 | 21 | 56 | 74 | 99 | 15 | 7 | 75 | 71 | 26 | 32 | 64 | 76 | 67 | 89 | 76 | 42 | 76 | 35 | 8 |
| 61 | 46 | 68 | 12 | 61 | 42 | 3 | 86 | 85 | 27 | 80 | 67 | 0 | 43 | 48 | 89 | 17 | 39 | 9 | 44 | 24 |
| 74 | 35 | 81 | 3 | 89 | 48 | 2 | 57 | 26 | 4 | 43 | 31 | 17 | 74 | 37 | 4 | 83 | 59 | 81 | 44 | 17 |
| 19 | 66 | 8 | 63 | 59 | 38 | 83 | 53 | 35 | 35 | 96 | 96 | 87 | 33 | 86 | 96 | 29 | 89 | 15 | 16 | 74 |
| 96 | 33 | 34 | 98 | 20 | 34 | 14 | 70 | 53 | 67 | 35 | 38 | 61 | 24 | 71 | 1 | 21 | 23 | 89 | 48 | 98 |

2 4 1 1 4 1 3 4 3 2

1 4 3 3 1 4 2 2 1 1

**Resolución**

*i* : Índice que recorre las 10 posiciones de la secuencia a generar

*j* : Índice que recorre las 4 líneas desde las que se pueden extraer unidades

*k* : Índice que recorre las unidades en cada una de las líneas

*rnd()* : Función que define un número aleatorio entre 0 y 1

*entero(x)* : Función que trunca un número aleatorio y se queda sólo con la parte entera

$x_i$ : Línea de la que extraer en la posición *i*

$a_{jk}$ : Vector que establece la unidad en posición *k* del producto *i*

a) Un algoritmo que genere una solución aleatoria podría tener la siguiente estructura.

```
For i:=1 to 10 do begin
    xi=entero(rnd()*4)+1;
end
```

Como ejemplo sean los números de dos cifras una secuencia aleatoria entre 0 y 99.

Dividiendo esos números por 100 tenemos números entre 0 y 1.

La secuencia de 10 números aleatorios es: 23 75 86 30 5 55 54 54 36 56

La secuencia de 10 líneas será: 1 4 4 2 1 3 3 3 2 3

b) Un algoritmo que genere una solución que podamos considerar correcta podría tener la siguiente estructura

```
for i:=1 to 10 do begin
    xi =Elegir la línea cuyo primer producto disponible dé un mejor valor de G(S*i)
end
```

Como ejemplo y dado que aquí no hay nada aleatorio, y dado que no tenemos valores para los productos en el almacén no es posible poner nada más que una secuencia de números del 1 al 4. Por ejemplo la anterior. La secuencia de 10 líneas será: 1 4 4 2 1 3 3 3 2 3

Otro algoritmo que genere una solución que podamos considerar correcta sería:

```
for i:=1 to 10 do begin
    xi =Elegir la línea cuyo primer producto disponible sea el más antiguo de todos.
end
```

Otro algoritmo que genere una solución que podamos considerar correcta sería:

```
for i:=1 to 10 do begin
        xi =(i mod 4) +1;
end
```

Este algoritmo daría como resultado 1,2,3,4,1,2,3,4,1,2 puesto La función mod es la función resto.

Otro algoritmo que genere una solución que podamos considerar correcta sería:

```
for i:=1 to 10 do begin
        xi =La línea con menor número de coches;
end
```

c) Un algoritmo de Mejora local exige definir un vecindario y un procedimiento de descenso. Un vecindario podría ser (poniendo un ejemplo diferente de los tradicionales) modificar el valor de uno de los elementos del vector solución. Por ejemplo en la solución anterior 1 4 4 2 1 3 3 3 2 3 posibles vecinos serían

2 4 4 2 1 3 3 3 2 3
3 4 4 2 1 3 3 3 2 3
4 4 4 2 1 3 3 3 2 3
1 1 4 2 1 3 3 3 2 3
1 2 4 2 1 3 3 3 2 3
1 3 4 2 1 3 3 3 2 3
1 4 1 2 1 3 3 3 2 3

Denominemos a este método *ModificarGen(i,j)* que significa que en la posición *i* probamos el efecto de la extracción de un unidad de la fila *j.*

Y el mecanismo de descenso puede ser el mecanismo de descenso rápido. Es decir en el momento que se encuentre algo mejor se sustituye

Una posible estructura del algoritmo sería:

```
generarSolucionVecinoMasCercano;
coste:=evaluarSolución(v);
i:=1;
while (i<=10) do begin
      j:=i;
      HayMejora:=false;
      while ((j<=4) and not(HayMejora)) do begin
                  j:=j+1;
            aux:=x(i)
            ModificarGen(i,j);
            if evaluarSolucion(v)<=coste then HayMejora:=true else mutaciónInversa(i,aux);
      end;
      if j=4 then i:=i+1;
      if HayMejora then i:=1;
end; // del while
```

d) Un algoritmo genético podría tener la siguiente estructura

```
generarPoblacionInicial;
while not(hayQueParar) do begin
      seleccionarPadres;
      construirNuevaSolucion;
      actualizarPoblación;
      provocarMutaciones;
end;
```

Por tanto para definir el procedimiento hace falta (asumiendo que tenemos una función G que evalúa)

Definir un tamaño de población inicial: Por ejemplo 10

Definir un modo de generación de la población inicial: aleatoriamente

Definir un modo de parar: al cabo de 1000 iteraciones.

Se selecciona un modo de selección de padres en función de una ruleta donde el valor de *fitness* sea la inversa de la valoración de la solución.

Se define un procedimiento de cruce: en este caso se propone un OX con un punto de cruce, donde la sección del Padre y la Madre se incorporan íntegramente no haciendo falta reparación.

Se actualiza la población: Si una solución obtenida del cruce es mejor que la peor de las soluciones la sustituye.

Para provocar mutaciones cada 100 iteraciones se sustituyen los 5 peores miembros de la población por 5 soluciones aleatoriamente construidas.

Un ejemplo para lo único representable (la función de cruce). Sean dos soluciones aleatorias construidas con los 10 primeros números de las filas segunda y tercera:

2 4 1 1 4 1 3 4 3 2

1 4 3 3 1 4 2 2 1 1

Se elige un número aleatorio (por ejemplo con el primero de la cuarta fila: 11 que dividido por 100 (para que esté entre 0 y 1) y multiplicado por 10 para elegir por donde cortar nos da 1,1 es decir 2. Cruzamos la primera y la segunda solución cortando en la segunda posición y da

2 4 3 3 1 4 2 2 1 1

e) Para modificar el algoritmo diseñado en b) teniendo en cuenta que en las líneas el número de unidades que puede haber puede ser muy bajo. Se propone lo siguiente:

*for i:=1 to 10 do begin*
*$x_i$ =Elegir la línea cuyo primer producto disponible dé un mejor valor de G(S*i) comprobando siempre que haya un producto disponible en dicha línea*
*end*

# 17 CASOS

## 17.1 Asignación de Fechas y Aulas para Exámenes

Una Escuela de Ingeniería ha experimentado una multiplicación en el número de títulos, alumnos y aulas en los últimos años. Este crecimiento ha supuesto una modificación sustancial en algunos de los procesos. Entre ellos el de asignación de aulas y fechas para exámenes.

Los diferentes problemas que se plantean por esta nueva circunstancia se pretenden resolver mediante una asignación automática al principio de curso.

Algunos de los datos del problema son:

| Número de Alumnos | Número de Asignaturas |
|---|---|
| 400-500 | 5 |
| 300-400 | 19 |
| 200-300 | 30 |
| 100-200 | 8 |
| 50-100 | 58 |
| 0-50 | 84 |

| Capacidad Aulas para exámenes | Número Aulas |
|---|---|
| 90 | 14 |
| 60 | 32 |
| | |
| | |
| | |
| | |

¿Puede establecer un método automático de asignación?

## 17.2 La Ruta de Llanes

En Arriondas (Asturias) usted tiene una de sus plantas envasadoras la Central Lechera “Carrete”. Esta central lechera, recoge leche de vaquerías situadas en distintas comarcas Asturianas y de las comunidades colindantes de la cornisa cantábrica española.

El sistema de recogida de leche hace tiempo que fue diseñado, y aunque ha habido variaciones importantes en las características de la recogida, hace tiempo que no se rediseña todo el proceso, sino que se hacen pequeños ajustes. La dirección pretende comprobar si compensaría hacer un estudio global, y para ello les requieren que analicen el proceso de recogida en la comarca de Llanes. Algunos datos relevantes son los siguientes:

1) La capacidad de los camiones que gasta nuestra empresa es de 20.000 litros.

2) La leche se guarda en enfriadoras durante un máximo de 2 días.

3) Cada vaquería tiene una capacidad productiva distinta.

4) Los costes de transporte se evalúan a razón de 50 pesetas el kilómetro realizado, aunque en realidad es un poco más caro ir cargado que de vacío.

5) La jornada laboral de cada transportista se considera de 40 horas semanales (con un margen “en negro” máximo de 15 horas).

6) Cada carga tiene un tiempo fijo de 15 minutos y una velocidad de carga de 200 litros por minuto.

7) Las velocidades de desplazamiento son de 60 kilómetros/hora por la N-634, 45 kilómetros/hora por la AS-114 y AS-115, y 35 kilómetros por hora en el resto de carreteras.

8) Por algunas carreteras la pendiente es tan acusada que no se puede descender con el camión cargado con más de 5000 litros, estás son:

a) La AS-340 en sentido Riensena-Nueva

b) La LLN14 Riensena-Palacio Meré en ese sentido

c) La LLN-7 Palacio Meré-La Pereda en ese sentido

Las vaquerías con las que se tiene concierto están en los siguientes pueblos (se incorpora además la capacidad de su enfriadora medida en litros, y su producción también medida en litros):

| | Nombre Pueblo | Enfriadora | Producción |
|---|---|---|---|
| 1 | Cangas de Onís | 15.000 | 6000 |
| 2 | Onís | 10.000 | 7000 |
| 3 | Riensena | 10.000 | 4000 |
| 4 | La Pereda | 10.000 | 7000 |
| 5 | Arenas de Cabrales | 10.000 | 3000 |
| 6 | Arangas | 20.000 | 4000 |
| 7 | Panes | 15.000 | 6000 |
| 8 | Noriega | 10.000 | 6000 |
| 9 | Colombres | 10.000 | 3000 |
| 10 | Cien | 10000 | 7000 |
| 11 | Carreña | 10000 | 5000 |
| 12 | Arrobio | 10000 | 4000 |
| 13 | Nava | 10000 | 5000 |
| 14 | Tazones | 10000 | 4000 |
| 15 | Tres Cares | 10000 | 3000 |

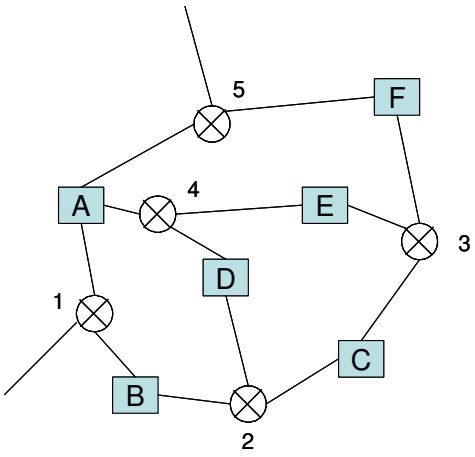
## *17.3 Sistema Eléctrico*

En un determinado sistema eléctrico se dispone de un número de centrales térmicas, nucleares e hidroeléctricas para afrontar una demanda que sigue en cada centro de transformación una función similar con respecto a cada hora del día. Esta relación se puede representar del siguiente modo:

| Hora del día | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| % sobre el consumo máx. diario previsto | 85 | 80 | 76 | 60 | 68 | 66 | 72 | 70 | 70 | 84 | 70 | 65 | 95 | 92 | 93 | 83 | 85 | 80 | 85 | 86 | 90 | 100 | 99 | 80 |

Donde la primera fila indica la hora del día considerada y la segunda el porcentaje sobre el consumo máximo diario previsto.

La red eléctrica tiene la siguiente configuración:




Donde todas las líneas tienen una capacidad de 500 MW excepto las que vienen reflejadas en la siguiente tabla:

| | |
|---|---|
| Línea B1 | 200 MW |
| Línea F5 | 400 MW |
| Línea E4 | 400 MW |
| Línea A4 | 600 MW |

Existen 5 Centros de transformación de alta tensión con demandas máximas según la siguiente tabla:

| Número del centro de transformación | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| Demanda Máxima (MW) | 250 | 1000 | 825 | 1150 | 700 |

El sistema descrito tiene 6 sistemas de generación: 3 son de tipo térmico y 2 de tipo hidráulico y uno de tipo nuclear. Los generadores térmicos y nucleares tienen que trabajar entre un máximo y un mínimo.

Se define un coste horario por trabajar al mínimo, más un coste extra por cada MW sobre el mínimo. Poner en marcha cada generador tiene también un coste fijo.

Los valores de dichos costes para las centrales térmicas vienen definidos en la tabla siguiente.

| | Nivel Mínimo | Nivel Máximo | Coste Mínimo | Coste Extra por MW | Coste de Puesta en Marcha |
|---|---|---|---|---|---|
| A | 40 % max | 1350 MW | 4000 € | 5 € | 3000 € |
| C | 50% max | 700 MW | 5000 € | 4.5 € | 2000 € |
| F | 40% max | 1050 MW | 9000 € | 5 € | 1000 € |

Además se dispone de dos centrales Hidroeléctricas, B y E. Éstas pueden trabajar a un nivel de producción fijo con un coste horario, y un coste fijo de puesta en marcha.

Una característica importante de estas centrales Hidroeléctricas es que el agua se puede subir otra vez al depósito desde un depósito inferior situado a los pies de la central.

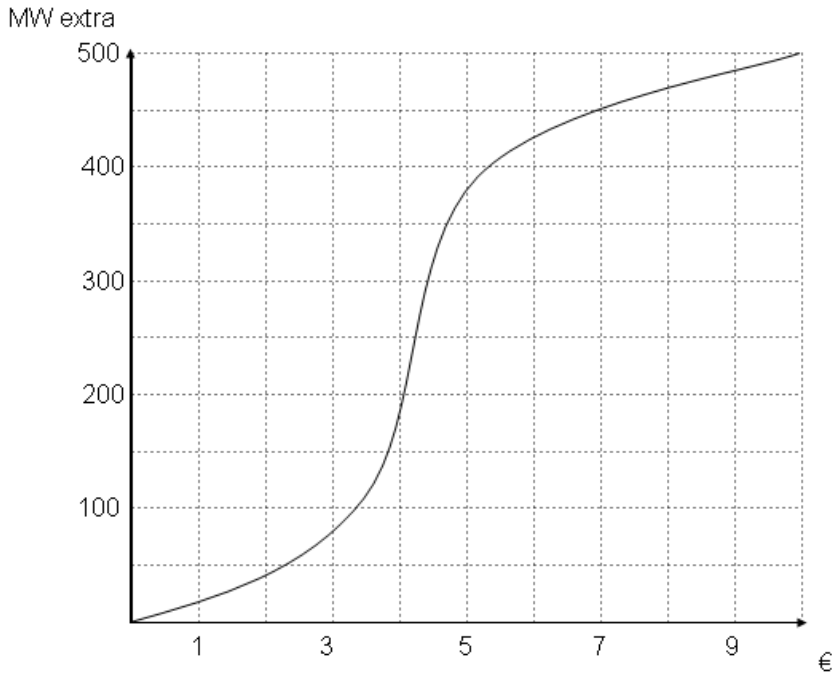
Por razones medio ambientales el depósito superior debe mantener siempre entre 10 y 20 metros de agua. Se supone que estamos en la estación seca, y no entra más agua en el depósito que la que ya hay.

| | Nivel de Generación | Coste por Hora | Reducción del nivel de agua en el depósito por hora | Coste de Puesta en Marcha |
|---|---|---|---|---|
| Central B | 450 MW | 350 € | 0,30 metros | 2000 € |
| Central E | 400 MW | 300 € | 0,48 metros | 2000 € |

Subir un metro de agua al depósito superior consume 3300 MW.

Por motivos de servicio, en cualquier momento debiera ser posible alcanzar un incremento de demanda sobre la prevista de un 15%. Este incremento se podría conseguir aumentando los niveles de trabajo de las centrales térmicas o poniendo en marcha las centrales hidroeléctricas, o utilizando la energía que se invierte en subir el agua al depósito superior, aunque ello suponga disminuir los 10 metros de agua citados.

La central nuclear tiene también unos costes máximos y unos mínimos, y sus costes extra por MW no son lineales y se representan por la siguiente curva:




| | Nivel Mínimo | Nivel Máximo | Coste Mínimo |
|---|---|---|---|
| D | 50 % max | 1000 MW | 4000 € |

Además los centros de transformación 1 y 5 pueden ser alimentados con más potencia comprando energía a una red eléctrica externa.

Se debe calcular la forma más barata de alimentar a la red eléctrica.

Calcular a partir de qué precios y en qué momentos del día es rentable comprar y vender energía eléctrica a las redes vecinas.

## *17.4 Optimización en la Refinería*

Una empresa que posee dos refinerías compra tres tipos de petróleo (PETR1, PETR2, PETR3) de diferentes calidades y por tanto de diferentes precios por barril. Dado que los planes de producción se establecen con un mes de antelación es necesario trabajar con previsiones de precio por barril. El departamento de compra establece los siguientes tres escenarios con las probabilidades expresadas en la tabla 1 y los precios en euros por barril para los próximos meses (por obvia simplificación del caso no se establecen probabilidades distintas para diferentes periodos)

| | PETR1 | PETR2 | PETR3 |
|---|---|---|---|
| Escenario 1 (35%) | 27 | 30 | 26 |
| Escenario 2 (35%) | 28 | 30 | 28 |
| Escenario 3 (30%) | 29 | 29 | 27 |

El petróleo se transforma a través de 4 procesos (Destilación, Reformado, Crackeado y Mezcla) para producir los productos que el consumidor normal compra: Aceite Lubricante, Gasóleo, Gasolina 95, Gasolina 98 y Keroseno.

**Destilación**

En la destilación se separa el petróleo en 6 fracciones (Nafta Ligera, Nafta Media, Nafta Pesada, Aceite Ligero, Aceite Pesado y Residuo). Las naftas ligeras, medias y pesadas tienen un octanaje medio de 95,85 y 75 respectivamente.

Las proporciones de cada fracción son (incluyendo las pérdidas):

| | Nafta Ligera | Nafta Media | Nafta Pesada | Aceite Ligero | Aceite Pesado | Residuo |
|---|---|---|---|---|---|---|
| PETR1 | 0.10 | 0.20 | 0.20 | 0.12 | 0.20 | 0.13 |
| PETR2 | 0.15 | 0.25 | 0.18 | 0.08 | 0.19 | 0.12 |
| PETR3 | 0.10 | 0.15 | 0.21 | 0.16 | 0.22 | 0.14 |

**Reforma**

Las Naftas se pueden usar directamente para mezclarlas dando lugar a gasolinas de diferentes octanajes o pasar por el proceso de reforma. Este proceso genera una gasolina denominada reformada con un octanaje de 125. En el proceso de reforma se obtiene:

- 0.6 barriles de gasolina reformada de 1 barril de nafta ligera
- 0.52 barriles de gasolina reformada de 1 barril de nafta media
- 0.45 barriles de gasolina reformada de 1 barril de nafta pasada

**Craqueado**

Los aceites (ligero y pesado) pueden ser usados para mezclar (dando lugar a *Gasóleo* y *Keroseno*) o se pueden pasar a través de un proceso conocido como *craqueado catalítico*. Este proceso produce aceite craqueado y gasolina craqueada. Ésta tiene un octanaje de 110.

- 1 barril de aceite ligero da lugar a 0.68 barriles de aceite craqueado y 0.28 de gasolina craqueada.
- 1 barril de aceite pesado da lugar a 0.75 barriles de aceite craqueado y 0.18 de gasolina craqueada.

La gasolina craqueada se utiliza para hacer gasolinas (95 y 98). El aceite craqueado se utiliza para hacer gasóleos (gasóleo y *Keroseno*).

Actualmente se está investigando en un producto químico que modificaría las anteriores proporciones:

- 1 barril de aceite ligero daría lugar a 0.63 barriles de aceite craqueado y 0.32 de gasolina craqueada.
- 1 barril de aceite pesado daría lugar a 0.77 barriles de aceite craqueado y 0.2 de gasolina craqueada.

Además, a partir de un barril de Residuo se puede obtener 0.6 barriles de Aceite Lubricante utilizando la capacidad del mismo sistema de craqueado.

**Mezclado**

Gasolinas

Hay dos tipos de gasolinas (súper y normal), que se obtienen por mezcla de naftas y gasolinas (craqueada y reformada). El octanaje de la gasolina súper debe ser superior a 98, y el de la gasolina normal a 95. Se asume que el octanaje se obtiene por combinación lineal de los volúmenes mezclados.

Keroseno

Se estipula que la presión de vapor del *Keroseno* debe ser inferior a 1 $Kg/cm^2$ . Las presiones de vapor, en las mismas unidades, de los productos que pueden dar lugar al *Keroseno* son los siguientes: Aceite Ligero (1), Aceite Pesado (0.6), Aceite craqueado (1.5) y Residuo (0.05). Se puede asumir otra vez que la presión de vapor resultado es combinación lineal considerando volúmenes.

Gasóleo

Para producir Gasóleo se utiliza aceite ligero, aceite craqueado, aceite pesado y residuo con una proporción 10:4:3:1.

Restricciones de capacidad

a) Disponibilidad diaria de los tres tipos de petróleo: 40.000, 60.000 y 40.000 barriles respectivamente

b) Capacidad de destilación: 40000 barriles diarios por refinería

c) Capacidad de reformado: 9000 barriles diarios por refinería

d) Capacidad de craqueado: 5000 barriles diarios por refinería

e) La fabricación de Gasolina 98 debe ser al menos el 30% de la Gasolina 95

Precios de Venta:

Los precios de venta de cada barril aportan los siguientes ingresos en Euros, descontados los gastos generales (de tal modo que los beneficios de la refinería se pueden calcular descontando estos ingresos de los costes de adquisición de los barriles).

| | Demanda | Beneficio Venta en refinería 1 | Beneficio Venta en refinería 2 |
|---|---|---|---|
| Aceite Lubricante | 4000 | 27.5 € | 28.5 € |
| Gasolina 95 | 26000 | 43 € | 50 € |
| Gasolina 98 | 10000 | 47 € | 45 € |
| Keroseno | 30000 | 39 € | 35 € |
| Gasóleo | 30000 | 34 € | 34 € |

Como se observa en la tabla los beneficios no son iguales para cualquier refinería debido a que no son plantas idénticas. Además en su construcción fueron diseñadas para que solo se pudiera trabajar con dos tipos de petróleo (por planta).

La demanda de cada producto tiene una cierta elasticidad con el precio que podemos considerar que se expresa del siguiente modo: “Una variación de un 1% en el precio implica una variación de sentido contrario en las ventas de un 5%”. Esta regla se mantiene para variaciones de precio inferiores al 10%.

## *17.5 Red de Metro de Valencia*

Trabaja usted en una consultora que ha sido contratada por la empresa FGV para realizar un programa que permita calcular el tiempo que se tarda en llegar entre dos paradas cualesquiera de la red de Metro Valencia.

Una vez desarrollada la herramienta les solicitan que hagan una exploración acerca de una nueva línea que se está planteando.


Dicha línea sería una nueva línea de tranvía que se engarzaría con la actual en las paradas de Empalme y La Carrasca, tal y como se muestra en el plano adjunto. Dicha línea tendría un punto medio en la parada de Jesús, donde se cruzaría con las Líneas 1 y 3, y con la Línea 5 en la Parada de Manuel Candela.

Los promotores de tan magno proyecto piensan que esta línea, además de aumentar la cantidad de población que puede tener contacto con la Red de Metro, aumentará la conectividad entre las distintas estaciones de la Red de Metro Valencia.

Otra interesante cuestión es saber si realmente es necesario cerrar la línea circular del Tranvía, para aumentar la conectividad o sobraría con fragmentos de la misma (por ejemplo de La Carrasca a Jesús).

## *17.6 Rutas de Distribución*

Una empresa de fabricación de cartones (cajas fundamentalmente) radicada en Alzira debe servir entre 4 y 7 camiones diarios a más de una veintena de clientes. Están interesados en una herramienta que permita definir la carga de los camiones teniendo en cuenta que estos tienen una limitación de capacidad de 33 paletas cada uno.

| PUEBLOS | Lote de Paletas |
|---|---|
| Alborache | 6 |
| Alcàcer | 2 |
| Algemesí | 2 |
| Almoines | 12 |
| Benifaió | 4 |
| Benigànim | 6 |
| Carcaixent | 2 |
| Carlet | 4 |
| Villanueva de Castellón | 6 |
| Chiva | 6 |
| Denia | 6 |
| L'Alcúdia de Carlet | 6 |
| L'Olleria | 12 |
| Llombai | 4 |
| Ontinyent | 12 |
| Silla | 4 |
| Sueca | 12 |
| Tavernes de la Valldigna | 4 |
| Xàtiva | 12 |

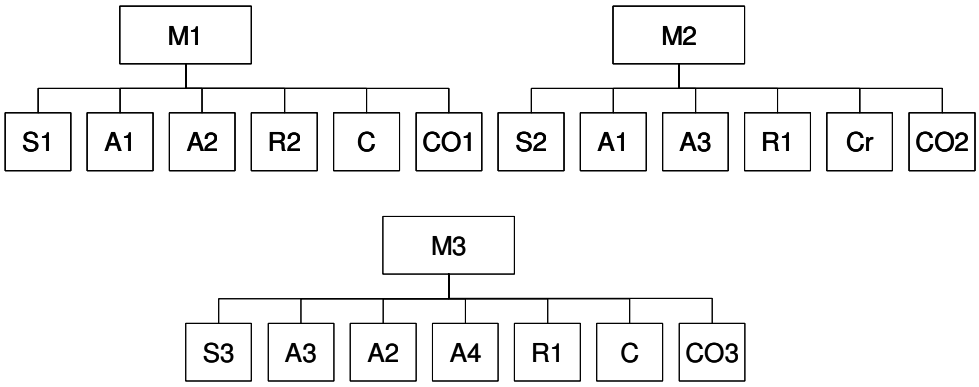
Se asume que en cargar un camión se tarda alrededor de 45 minutos, la descarga en cada empresa cliente cuesta alrededor de 30 minutos de media. Se asume que ningún conductor debe dedicar más de 8 horas diarias a conducir ni más de 12 horas en total. Cada kilómetro cuesta 30 céntimos de Euro.

## *17.7 Fabricación de Zapatillas*

Una empresa fabricante de zapatillas deportivas, localizada en Picasent, estima la siguiente demanda (en miles de pares) para los próximos 6 meses, con un catálogo inicial a la venta de 3 modelos.

| | Abril | Mayo | Junio | Julio | Agosto | Septiembre |
|---|---|---|---|---|---|---|
| Modelo 1 | 100 | 200 | 150 | 150 | 200 | 150 |
| Modelo 2 | 150 | 100 | 150 | 100 | 250 | 100 |
| Modelo 3 | 150 | 150 | 250 | 150 | 150 | 150 |

Se admite un error en la previsión de demanda de cada modelo de 3 miles de pares.




Los elementos que se utilizan en la producción de estos 3 modelos de zapatillas son los siguientes: Cordones (C), Cremallera (Cr), Suela (tipos: S1, S2, S3), Refuerzos (tipos: R1, R2), Cuerpo (tipos CO1, CO2, CO3) y adornos (A1, A2, A3, A4).

Como se puede observar, entre los diferentes modelos de zapatillas a producir se da la particularidad que comparten componentes entre un modelo y otro. Las suelas tienen un periodo de aprovisionamiento de 4 semanas. Los refuerzos y el cuerpo un periodo de aprovisionamiento de 2 semanas, los adornos de 1 semana, y las cremalleras de un plazo de 8 semanas. Se dispone en almacén de 8000 de estas cremalleras. Se dispone también en almacén de suficientes componentes para cubrir la demanda de los próximos dos meses, de los diferentes componentes. Los pedidos se lanzarán en miles de unidades, a excepción de las cremalleras que se lanzaran en lotes de cinco mil unidades.

Los costes de producción de cada par se evalúan en 7€ si se producen en horas normales, y 9€ si se produce en horas extras. El número de horas normales disponibles por día son 8. El número de días laborables por mes es variable cada mes, dependiendo del calendario laboral de la localidad de Picassent. Cada mes se puede trabajar un máximo de 40 horas extra. Un grupo de 3 personas es capaz de fabricar 15 pares de zapatillas por hora, si dispone de suficiente material. El coste de almacenar un par de una semana para otra es de 0’025 € por par. Se dispone de 200 pares en stock en estos momentos, de cada uno de los modelos.

Así, se pretende establecer el modelo que permitiera definir el plan de trabajo para los próximos meses, asegurando el mínimo coste para la empresa. Al ingeniero se le ha planteado un compromiso con los trabajadores de utilizar cada dos meses al menos un 15% de la capacidad en horas extra actual, por lo que debería plantearse como programar también esa circunstancia. Además es habitual que en el mes de Agosto, no se trabaje la mitad del mes, y que los trabajadores de la empresa cojan los 15 días restantes durante los meses de Junio, Julio, Agosto o Septiembre dependiendo de la disponibilidad de la empresa.

Durante las vacaciones de verano se debe considerar la contratación de trabajadores para sustituir a los trabajadores que están todo el año. Los costes de producción de cada par se evalúan en 8€ en horas normales no pudiendo producir estos trabajadores en horas extras. El ritmo de producción de estos trabajadores es de 18 pares de zapatillas por hora. Se podrá contratar uno o dos grupos.

Se plantean diferentes alternativas al definir el objetivo principal. Así, una de ellas podría ser el minimizar el máximo stock entre periodos. Otra, minimizar las diferencias de producción entre un mes y el siguiente, para cualquier mes, a través de equilibrado de la producción utilizando horas extras, minimizar el stock en almacenes...

## *17.8 Las Farmacias de Alcudia, Benimodo y Carlet*

Está empezando una nueva carrera de consultor *free-lance* y por tanto acepta cualquier tipo de trabajo. Le ha llegado un trabajo por vía de un amigo (el único modo de que entren trabajos). El trabajo no parece estar muy definido.

*Hola,*

*Me ha dado tu dirección Moisés de la Fuente. Soy Marina Amorós, farmacéutica de Carlet y estoy integrada con las farmacias de mi pueblo y de otros 2 cercanos, para hacer las guardias. En total, somos 11 farmacias de tres pueblos cercanos:*

- *5 de Carlet*
- *5 de L´Alcudia*
- *1 de Benimodo*

*Entre las 11 farmacias nos turnamos los turnos de guardia (noche) alternativamente de forma que "cada pueblo no esté más de 1 noche sin tener una farmacia abierta, excepto cuando toca Benimodo que no queda más remedio que sean 2 noches". Tenemos tres turnos:*

1. *Días laborales*
2. *Sábados*
3. *Domingos y festivos*

*Incluso en la última reunión que tuvimos este mes pasado se planteó incluso hacer otro turno (el cuarto ¡!) que incluyera el 24 dic (Nochebuena), el 31 de dic y el 5 de Enero.*

*Muchos estamos hartos de tanto turno porque hay semanas que nos coinciden 2 o 3 guardias, sobre todo en puentes y preferimos un sólo turno cada 11 días (es lo más habitual entre los pueblos que nos rodean). Parte de nuestros compañeros dicen que no es equitativo que siempre hay alguno que le tocan más domingos que otro, etc. Mi pregunta es si sabes alguna fórmula matemática para hacerlo más equitativo*.

Puesto manos a la obra usted le envía el siguiente mensaje:

*Hola Marina, más que tener dudas de los turnos, lo que me gustaría saber son las fechas que se consideran festivos en cada una de las localidades, porque supongo que no serán iguales. Un turno especial de Navidad no creo que sea práctico, tardaríais 11 años en darle la vuelta.*

*Mi propuesta es plantear un sistema que permita establecer una secuencia donde se establezcan tantas restricciones como se quieran (siempre que sea matemáticamente posible). Pero no tendréis un ciclo de 11 días sino más bien, un calendario anual (o semestral o bianual) que intenta equilibrar las cargas. Se me ocurre investigar un modo de establecer el calendario respetando restricciones de no más de 1 guardia en 6 o 7 días, no más de un festivo en 15 días… Te adjunto una tabla en Excel® que puedes completar con los festivos que faltan. Como hay once columnas todo consistiría en sortear cada columna a cada persona.*

*PD: Por cierto, ¿Qué ocurre cuando un sábado coincide con festivo?, ¿Qué manda?*

La respuesta, que obvia los datos de las festividades, es la siguiente

*Buenos días,*

*Ante todo, muchísimas gracias por el interés. Respondiendo a tu pregunta, cuando un sábado coincide con un festivo manda siempre el festivo, por tanto no hacemos refuerzos de sábado (cada pueblo tiene todos los sábados una farmacia abierta hasta las 5).*

*Gracias de nuevo.*

El trabajo consiste en:

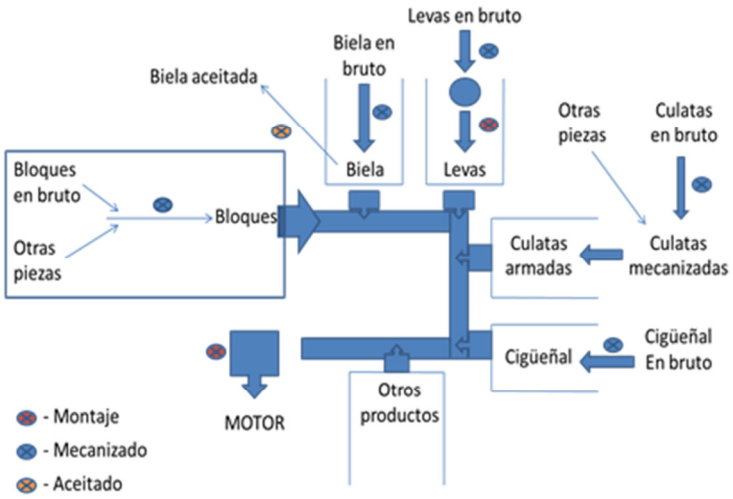
a) Definir de modo estructurado el problema
b) Diseñar un Modelo matemático que lo resuelva
c) Implementar el modelo en MPL, para resolverlo
d) Diseñar una heurística, o conjunto de ellas, que resuelvan satisfactoriamente el problema

## 17.9 Planificación Agregada en una Planta de Motores

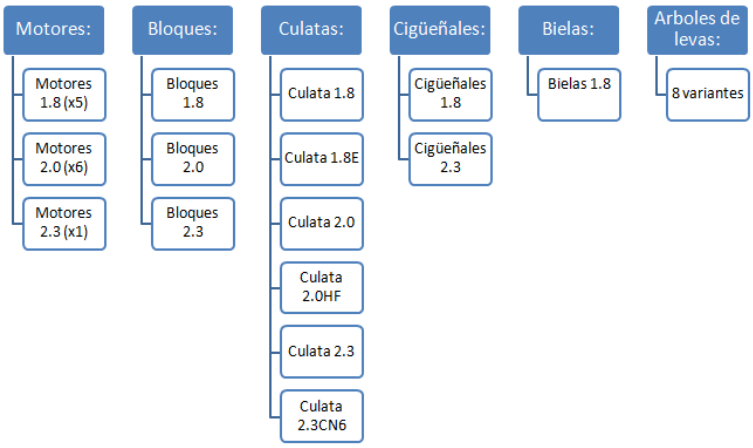
Una empresa de fabricación y montaje de motores sirve motores y componentes de los mismos a plantas de fabricación de coches así como a empresas de distribución de repuestos. Ambos tipos de clientes están distribuidos por todo el mundo. Al ser los clientes distintos los modos de preparación del producto y embarque del mismo pueden variar entre clientes. No sólo venden producto final sino también semielaborado. Y también compran semielaborado. Asimismo diferentes clientes y diferentes proveedores tienen necesidades y calendarios variables.

En un mercado tan competitivo como el del automóvil las actualizaciones de los productos son frecuentes. Además de que el número de variantes a producir tiende a crecer para adecuarse o superar a la competencia. El número de productos a fabricar es siempre limitado y conocido, aunque este conjunto varía. Pero varía de un modo planificado. Asimismo Esto implica pruebas de productos y nuevas versiones y también implica la desaparición planificada de productos.

El proceso de fabricación de un motor tiene dos etapas: a) Fabricación y b) Montaje. En la Fabricación un elemento en bruto es mecanizado, y en el montaje diferentes elementos mecanizados se ensamblan en una línea de montaje.




Un motor se compone de las denominadas 5C's (Bloque, Bielas, Levas, Culatas, Cigueñales) y otros componentes. De cada tipo de componente hay un cierto número de variantes, algunas se fabrican en la planta objeto de estudio y otros no.




La línea de Montaje de una fábrica de Motores es un sistema altamente automatizado, bastante poco flexible en cuanto a capacidad, y costoso de puesta en marcha y mantenimiento. Dentro de la escasa flexibilidad reconocida a la línea existe la posibilidad de pararla o hacerla funcionar a diferentes velocidades con costes diferentes para cada una de las opciones.

Aunque el proceso de fabricación es diferente lo mismo ocurre, con diferente dimensión, en las líneas de mecanizado de las conocidas como 5C's.

La empresa planifica en semanas y cada semana tiene una serie de días hábiles (5 por lo general) y de inhábiles (2 por lo general) pero la empresa podría decidir parar un día la producción pues ahorraría costes de energía, o utilizar uno de los días inhábiles. La empresa utiliza en ocasiones las horas extras pero estas no se planifican puesto que se utilizan para responder a problemas puntuales.

En este sentido el funcionamiento de la planta es atípico tanto al nivel de la gestión de la producción como en lo referente a la consideración del factor humano. Los modos de gestión de la capacidad son dos: Incrementar o reducir la velocidad de la línea (incorporando nuevos recursos a la misma), y parar/utilizar la línea en días laborales/días festivos.

En efecto, a causa del nivel de automatización de la línea y de los costes que implica el hacer funcionar la maquinaria, se autoriza el paro total de la planta cuando sea necesario, ya que se sabe que el coste de mano de obra es muy bajo en comparación con el de la mantener en funcionamiento la maquinaria. En este caso, los operarios van a formación o se dedican a tareas de mantenimiento y se ahorran los gastos de hacer funcionar la línea. Así pues, consideraremos que los directivos pueden decidir que hay que trabajar unos días festivos por cualquier razón. Además si la línea no tiene capacidad de cumplir la demanda durante la semana normal, es decir entre el lunes y el viernes, el modelo deberá ser capaz de proponer la cantidad de días extras que se necesitan para evitar diferir (dentro de los límites impuestos por la empresa durante cada semana).

Los inventarios no tienen un coste de mantenimiento conocido, aunque este es elevado dadas las características de los productos. Para mantener bajo control los inventarios la empresa prefiere establecer un stock mínimo y máximo (medido en días de stock) para el final del horizonte de planificación. Y dicho horizonte de planificación se sitúa siempre al final de un periodo de vacaciones

relativamente prolongado (verano, pascua, navidad) de tal modo que no se tendrá más stock del deseado en ningún momento sin necesidad de asignarle un coste.

## *17.10 Karbonatadas JUPE (III)*

Una empresa que fabrica refrescos carbonatados tiene una fábrica con capacidad de envasar 70 hectólitros a la hora.

La fábrica puede trabajar a 1,2,3 y 4 turnos de 80 horas semanales cada uno (deduciendo la parte proporcional de festivos no fin de semana). Es decir la capacidad productiva de una determinada semana se calcula según la siguiente fórmula

$$40 \cdot K \frac{(5-f)}{5}$$

Siendo 40 el número de horas laborables por semana en un turno completo, K la capacidad de envasado medida en hectólitros por hora, y f el número de festivos en una semana (considerando sólo los festivos que reducen el número de días laborables).

Están en las últimas semanas del año, y el departamento de finanzas se ha puesto a trabajar, solicitando presupuestos. Usted debe anticipar el plan de producción para poder aportar información al departamento de Recursos Humanos y al de Logística.

Ventas le anticipa la demanda en hectólitros en forma de 4 escenarios posibles, con la demanda prevista en centenares hectólitros por semana (tabla 1).

El departamento de RRHH le ha pedido un presupuesto para su gasto del año que viene. Un turno tiene un coste semanal de 6000 euros. Cada cambio de configuración del número de turnos, por ejemplo pasar en una semana de 2 turnos a 3 turnos, tiene un coste de 1000 euros por turno cambiado. El departamento de recursos humanos tiene cerrado ya el programa de trabajo de las primeras 5 semanas del año y debe cerrar las de la semana de 6 a 10.

| | 03/01/2011 | 10/01/2011 | 17/01/2011 | 24/01/2011 | 31/01/2011 |
|---|---|---|---|---|---|
| Turnos | 1 | 1 | 1 | 2 | 2 |

El departamento financiero le ha pedido un presupuesto para su gasto en almacenes subcontratados este año.

El departamento de logística tiene un almacén propio con una capacidad de 8000 hectólitros. Si los productos se guardan en el almacén propio el coste de almacenar cada hectólitro se evalúa en alrededor de 0,25 € por semana. Dispone también de almacenes a los que puede subcontratar capacidad. si los productos se guardan en almacén ajeno el coste de almacenar (que incluye el coste de transportar) se eleva a 1 € por semana. Al principio de año calcula que tendrá 5000 hectólitros de stock.

Se le pide que :

a) Diseñe un modelo matemático para establecer el programa óptimo teniendo en cuenta los 4 escenarios.
b) Defina los costes en recursos humanos y en almacenes en los que incurrirá.

## *17.11 Central Pendiente Dominicana*

Central Pendiente Dominicana CPD, es una floreciente cadena de supermercados. Actualmente cuenta con 14 tiendas de diferentes tamaños, ubicadas geográficamente en los puntos que se reflejan en la tabla 1. El consumo de cada una de las tiendas se estima en peso. Y la demanda debe ser satisfecha.

| Tiendas | x | y | Demanda (€) |
|---|---|---|---|
| Tienda1 | 10 | 40 | 1000 |
| Tienda2 | 30 | 50 | 2000 |
| Tienda3 | 20 | 50 | 1000 |
| Tienda4 | 40 | 10 | 500 |
| Tienda5 | 50 | 30 | 600 |
| Tienda6 | 50 | 20 | 2000 |
| Tienda7 | 40 | 40 | 700 |
| Tienda8 | 50 | 50 | 800 |
| Tienda9 | 50 | 30 | 1200 |
| TiendaA | 60 | 40 | 900 |
| TiendaB | 100 | 20 | 700 |
| TiendaC | 12 | 90 | 600 |
| TiendaD | 90 | 90 | 1000 |

Tabla 1. Ubicación y Demanda de cada Tienda

CPD cuenta con 4 almacenes, ubicados en los puntos de la geografía que se indican en la tabla 2. Dichos almacenes están mayormente situados cerca de sus mayores puntos de consumo. Actualmente está considerando la posibilidad de abrir un nuevo centro logístico, más grande y alejado. Los datos básicos se representan en la siguiente tabla.

| Almacenes | X | Y | Capacidad(€) | C Fijo | C Variable por € |
|---|---|---|---|---|---|
| Almacen1 | 10 | 20 | 5000 | 1000 | 1 |
| Almacen2 | 14 | 30 | 5000 | 1000 | 1 |
| Almacen3 | 23 | 60 | 5000 | 1000 | 1 |
| Almacen4 | 50 | 50 | 5000 | 1000 | 1 |
| Almacen5 | 200 | 200 | 20000 | 2000 | 0,1 |

Tabla 2. Ubicación, Capacidad y Coste de Cada almacén

Usted está analizando la cadena de suministro de productos perecederos, y CPD cuenta con tres fábricas de envasado. Sus características están en la tabla 3.

| Fabricas | X | Y | Capacidad(€) | C Fijo | C Variable por € |
|---|---|---|---|---|---|
| Fábrica1 | 10 | 10 | 10000 | 500 | 5 |
| Fábrica2 | 30 | 50 | 10000 | 700 | 2 |
| Fábrica3 | 40 | 40 | 10000 | 1000 | 1 |

Tabla 3. Ubicación, Capacidad y Coste de Cada Fábrica

Los centros de aprovisionamiento de los productos que vende están fundamentalmente en dos puntos de la geografía que se indican en la tabla 4.

| Abastecimiento | X | Y |
|---|---|---|
| Proveedor1 | 0 | 0 |
| Proveedor2 | 70 | 20 |

Tabla 3. Ubicación del centro de gravedad de los puntos de abastecimiento

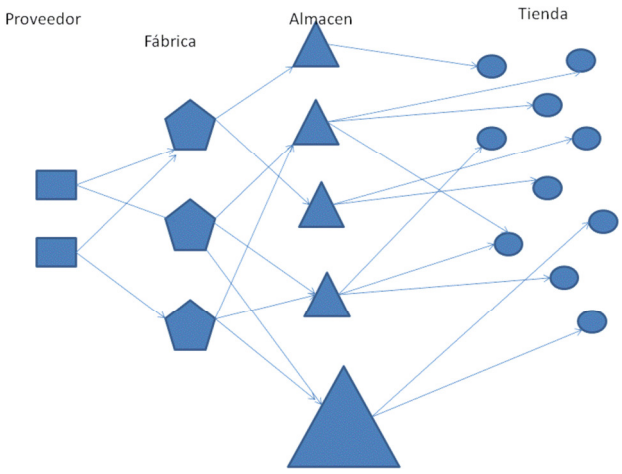
Los costes de transporte se calculan en pesos por kilómetro recorrido y m3 transportado.

Los costes de almacenamiento y de procesamiento del producto se calculan teniendo en cuenta un coste fijo (relacionado con poner en marcha la instalación) y un coste variable medido en pesos por m3 movido.

| | | |
|---|---|---|
| coste del km | 1 | €/$m^3$ transportado/km |
| coste de almacenaje | 1 | €/$m^3$ almacenado/día |
| Valor del $m^3$ | 10 | €/$m^3$ |
| coste de inventario | 0,05 | €/€ almacenado/día |

¿para qué densidad de valor (€/m3) del producto es rentable irse al almacén grande único y alejado?

Construya un modelo matemático



ETAPA 1:

Una empresa tiene 3 fábricas y 4 almacenes.

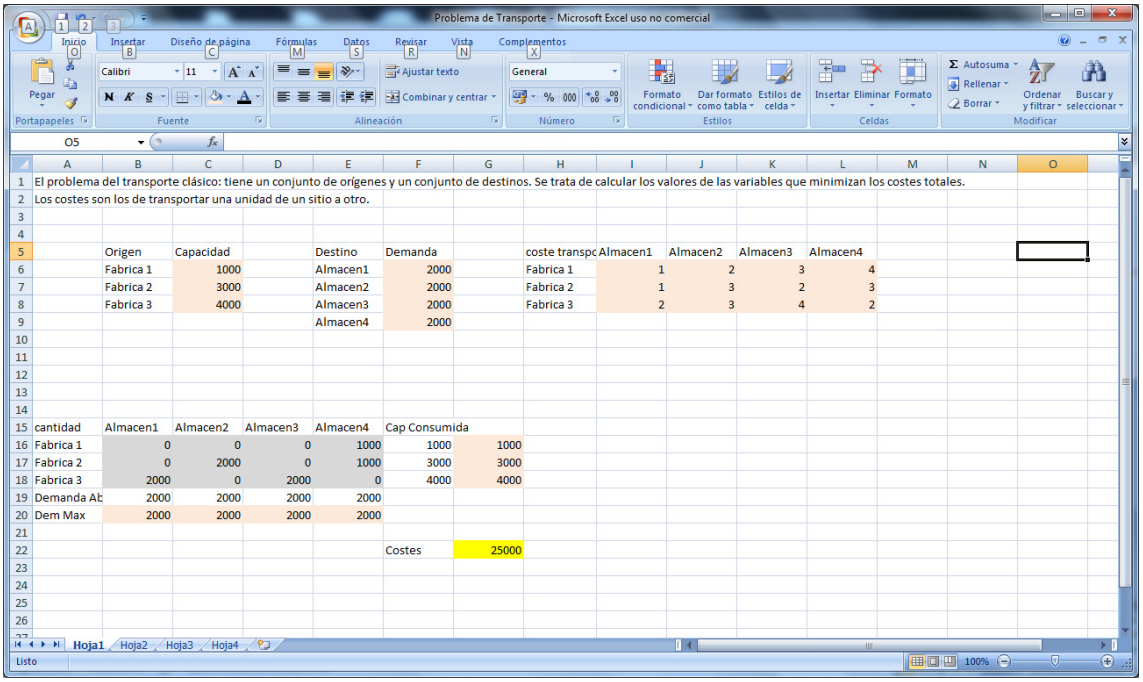
La capacidad de cada fábrica es {10000;4000;4000}

La demanda a servir a cada almacén es {2000;2000;2000;2000}

Los costes de transportar desde cada fábrica a cada almacén son:

| | Almacen1 | Almacen2 | Almacen3 | Almacen4 |
|---|---|---|---|---|
| Fabrica 1 | 1 | 2 | 3 | 4 |
| Fabrica 2 | 1 | 3 | 2 | 3 |
| Fabrica 3 | 2 | 3 | 4 | 2 |

¿Quién debe servir a quien para tener el mínimo coste?




Utilice el *solver* para resolver.

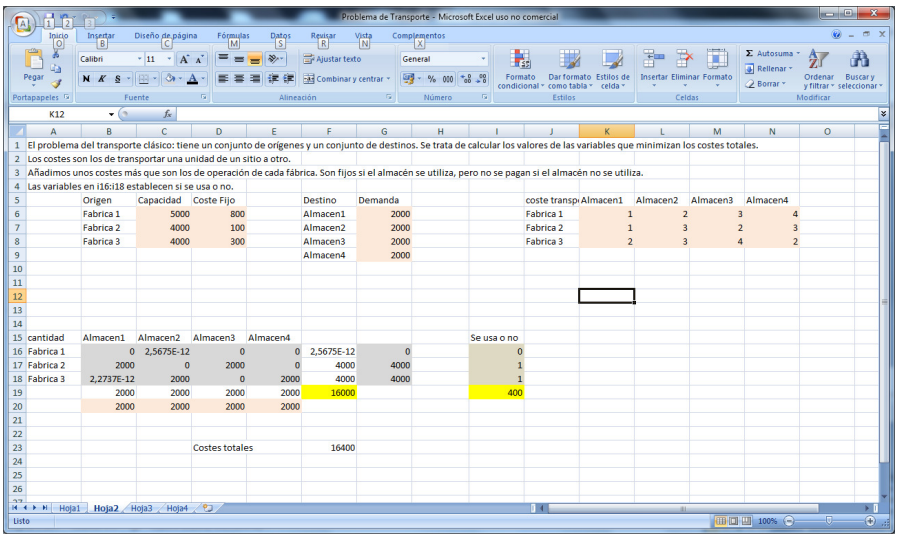
¿Es mejorable la solución que le entrega?

¿Cómo afectan los costes de transporte a los costes totales?

**ETAPA 2: Sobre el problema anterior.**

Cada fábrica tiene que pagar un coste fijo por ser utilizada.

El coste de cada fábrica son {800;1000;1200}



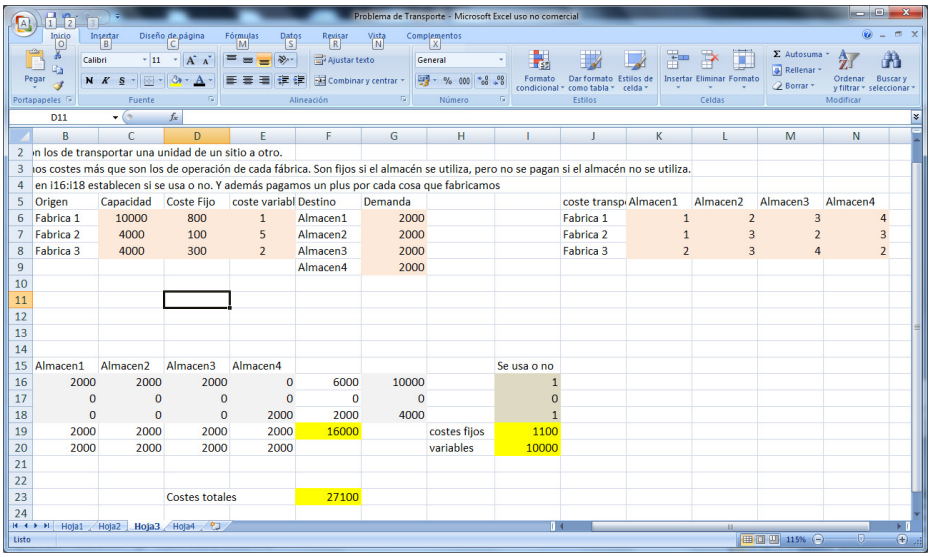
Utilice el *solver* para resolver.

¿Por qué cree que deben existir esos costes fijos?

**ETAPA 3: Sobre el problema anterior**

Cada fábrica tiene que pagar un coste variable por cada unidad que utiliza.

El coste unitario de producir en cada fábrica son {3;2;1}



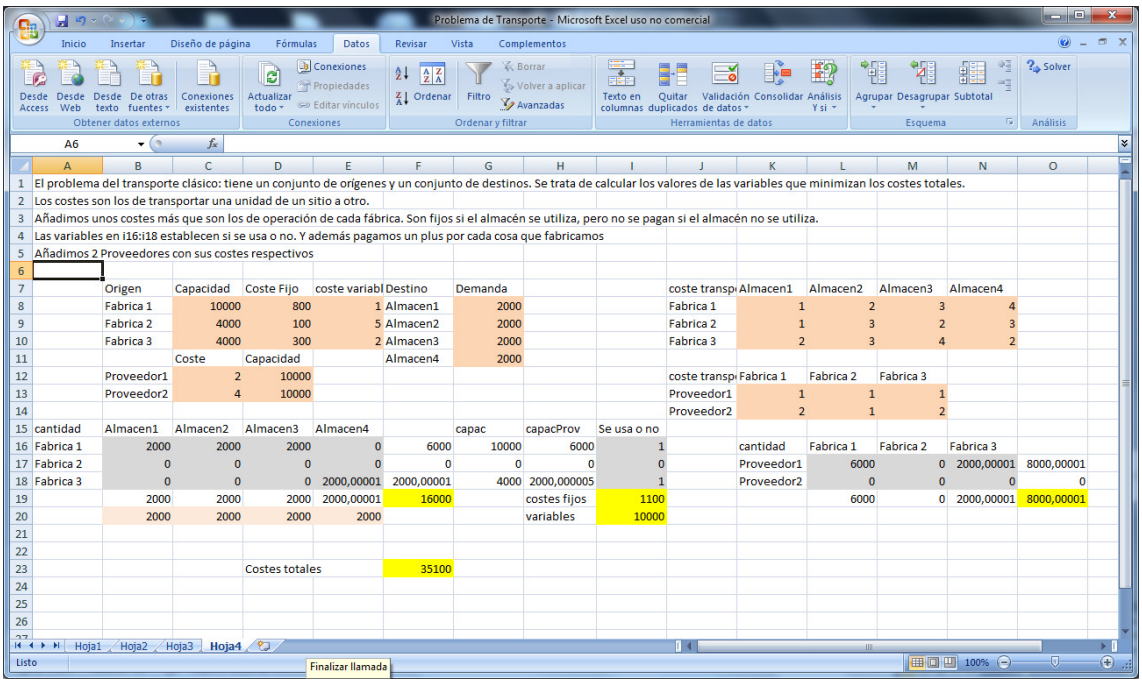
Utilice el *solver* para resolver.

¿Por qué cree que deben existir esos costes variables? ¿Cree que deben ser constantes o dependen según la cantidad?

**ETAPA 4**

Añadimos dos posibles proveedores, con sus costes de aprovisionar a cada fábrica respectivamente.

Las fábricas sólo pueden fabricar si tienen materia prima suficiente.




Grafíe como evolucionaría la red completa a medida que va subiendo el valor de euros por metro cúbico transportado.

# 18 BIBLIOGRAFÍA


Bautista, J., Companys, R., Corominas, A. Introducción al BDP. 1992.

Binmore, K. Teoría de Juegos. McGraw-Hill.1994

Bramel, J. D. Simchi-Levy. The Logic of Logistics. Springer. 1997

Chase, R.B. N.J. Aquilano, F.R. Jacobs. Administración de producción y operaciones. Irwin-McGraw-Hill. 2000

Chrétienne, P. Coffman, E.G., J.K. Lenstra, Z. Liu. Scheduling Theory and its Applications. Wiley and Sons. 1997

Companys, R., 2002. Programacion dinamica. CPDA-ETSEIB.

Companys, R., 2003. Teoria de Grafos. cpda-etseib.

Conway, R.W., W.L. Maxwell, L.W. Miller. Theory of Scheduling. Addison-Wesley. 1967

Cuatrecasas, Ll.. Organización de la Producción y Dirección de Operaciones.

Dembo, R.S.. Scenario Optimization. Annals of Operations Re*search* 30, 63-80. 1990

Denardo, E.V., 1982. Dynamic Programming. Models and Applications. Prentice Hall.

Díaz, A., F. Glover, H. Ghaziri, J.L. González, M. Laguna, P. Moscato, F.T. Optimización Heurística y Redes Neuronales. 1996

Diaz, A., Glover, F., Ghaziri, H., Gonzales, J.L., Laguna, M., Moscato, F.T., 1996. Optimización Heurística y Redes Neuronales. McGriwHall.

Ferreira, A. M. Morvan. Models for parallel algorithm designs: an introduction. Parallel computing in Optimization. Kluwer Academic Publishers, 1997.

Ferreira, P, A.. Pardalos. Solving Combinatorial Problems in Parallel. Springer. 1996

Gendreau, M., 2003. An introduction to tabu *search*. In: Glover, F., Kochenberger, G.A. (Eds.), Handbook of metaheuristics Kluwer International Series, pp. 37-54.

Glover, F., Laguna, M., Marti, R., 2003. *Scatter Search* and Path relinking: advances and applications. In: Glover, F., Kochenberger, G.A. (Eds.), Handbook of metaheuristics Kluwer International Series, pp. 1-35.

Gross, D., Harris, C.M., 1998. Fundamentals of Queueing Theory. John Wiley and Sons.

Gross, D., Shortle, J.F., Thomson, J.M., Harris, C.M., 2008. Fundamentals of Queueing Theory. Wiley.

Hansen, P., Mladenovic, N., 2003. *Variable Neighborhood Search*. In: Glover, F., Kochenberger, G.A. (Eds.), Handbook of metaheuristics Kluwer International Series, pp. 145-184.

Henderson, D., Jacobson, S.H., Jonhson, A., 2003. The theory and practice of *Simulated Annealing*. In: Glover, F., Kochenberger, G.A. (Eds.), Handbook of metaheuristics Kluwer International Series, pp. 287-319.

Johnson, L.A., D.C. Montgomery. Operations Re*search* in Production Planning, Scheduling and Inventory Control. John Wiley and Sons. 1974

Kauffman, A., 1972. Métodos y Modelos de la Investigación de Operaciones. CECSA.

Kauffman, S., 2003. Investigaciones. Metatemas.

Kauffmann A., Henry-Labordere, A.. Métodos y Modelos de la Investigación de Operaciones Tomo III. CECSA. 1976

Kauffmann, A.. Métodos y Modelos de la Investigación de Operaciones Tomo II. CECSA. 1972

Lourenço, H.R., Martin, O.C., Stützle, T., 2003. *Iterated Local Search*. In: Glover, F., Kochenberger, G.A. (Eds.), Handbook of metaheuristics Kluwer International Series, pp. 321-353.

Marti, R., 2003. Multistart Mehods. In: Glover, F., Kochenberger, G.A. (Eds.), Handbook of metaheuristics Kluwer International Series, pp. 355-368.

Osman, I.H., 1995. An Introduction to Metahuristics. In: Lawrence, M., Wilsdon, C. (Eds.), Operational Re*search* Tutorial Papers. Stockton Press, Hampshire (UK), pp. 92-122.

Perez, J., Mladenovic, N., Batista, B., Amo, I., 2006. Variable Neighbourhood *Search*. In: Alba, E., Mart+¡, R. (Eds.), Metaheuristic Procedures for Training Neutral Networks Springer US, pp. 71-86.

Pidd, M., 1996. Tools for thinking. Wiley.

Pinedo, M., Chao X.. Operations Scheduling. McGraw-Hill. 1999

Raiffa, J.F. Decision Analysis. Addison Wesley. 1970

Resende, M.G., Ribeiro, C.C., 2003. *Greedy* Randomized Adaptive *Search* Procedures. In: Glover, F., Kochenberger, G.A. (Eds.), Handbook of metaheuristics Kluwer International Series, pp. 219-249.

Shapiro, J.F. Modelling the Supply Chain. Duxbury. 2001

White, D.J. Teoría de la Decisión. Alianza Universidad. 1972

Williams, H.P. Model Building in Mathematical Programming. Wiley. 1999