UNIVERSIDAD PONTIFICIA COMILLAS

ESCUELA TECNICA SUPERIOR DE INGENIEROS INDUSTRIALES

EL PROBLEMA GENERAL DE LA OPTIMIZACION DE DISEÑO POR ORDENADOR: APLICACION DE TECNICAS DE INGENIERIA DEL CONOCIMIENTO

TESIS DOCTORAL

Fernando de Cuadra García

Ingeniero Industrial por la E.T.S.I.I de la

Universidad Pontificia Comillas

1990

TESIS DOCTORAL

EL PROBLEMA GENERAL DE LA OPTIMIZACION DE DISEÑO POR ORDENADOR: APLICACION DE TECNICAS DE INGENIERIA DEL CONOCIMIENTO

Por: D. Fernando de Cuadra García

Director de la Tesis: D. José Ignacio Pérez Arriaga

TRIBUNAL CALIFICADOR:

D. José Cuena Bartolomé

D. Gregorio Fernández Fernández

D. Luis García Pascual

D. José Ignacio Pérez Arriaga

Dña. Carme Torras i Genís

Madrid, Octubre 1990

A mis padres, Ignacio y Ana María.

A mi amigo Juan Carlos Lavalle.

AGRADECIMIENTOS

Deseo expresar mi más profundo agradecimiento a cuantas personas e instituciones han hecho posible la realización de esta tesis doctoral:

En primer lugar a mi director de tesis y amigo D. Jose Ignacio Pérez Arriaga, por su extraordinaria labor de dirección, documentación y apoyo; es difícil imaginar a alguien tan brillante y que muestre a la vez tal preparación y entrega en su trabajo.

A mis padres, por animarme constantemente a seguir el camino de la investigación. Por su ayuda y su constante aliento, a mi profesor y amigo D. Jorge Rodríguez-Piñero y a todos mis compañeros del IIT, y muy especialmente a D. Juan Carlos Lavalle Núñez, a D. Juan José Alba Ríos y a D. Juan José San Gil Otero. A D. Juan Rivier Abbad, por su eficacia y esfuerzo en el desarrollo de la herramienta informática ANA.

A Electrotécnica Arteche Hermanos S.A., y en particular a D. Jaime Berrosteguieta y D. Angel Enzunza, por su colaboración en los proyectos de los cuales ha surgido esta tesis, así como a D. Marc Toussaint, D. Giorgio Solari, D. Joop de Cruiff (European Space Agency), al profesor Jeffrey H. Lang (LEES, MIT) y a D.Juan Jesús León Cobos (GMV S.A.)

Finalmente quiero agradecer al Instituto de Investigación Tecnológica y al Ministerio de Educación y Ciencia su soporte económico durante los años de desarrollo de la tesis.

Madrid, a 27 de Septiembre de 1990

EL PROBLEMA GENERAL DE LA OPTIMIZACION DE DISEÑO POR ORDENADOR: APLICACION DE TECNICAS DE INGENIERIA DEL CONOCIMIENTO.

Capítulo 1

Introducción

El objeto de esta tesis es presentar un método sistemático, integral y general para desarrollar herramientas de optimización de diseño por ordenador. Este método se considera general por ser aplicable al diseño de una amplia gama de máquinas y sistemas, e incluso a problemas de optimización de planes. Se considera integral por abarcar desde el establecimiento teórico y formal del problema del diseño hasta la implantación informática, pasando por la extracción y representación del conocimiento. Por último, es sistemático en cuanto que establece las fases de desarrollo de un proyecto genérico, los objetivos de cada una de estas fases y los criterios generales para alcanzar dichos objetivos.

Las herramientas de diseño por ordenador tradicionales (CAD) sirven para evaluar un diseño ya fijado por el usuario, o al menos para facilitar la evaluación de un sistema mediante un buen interfaz. Tal es el caso de los programas que representan objetos en dos o tres dimensiones, comprueban el comportamiento de circuitos eléctricos o estudian esfuerzos en estructuras complejas. Es el usuario quien hace los cambios en el diseño de acuerdo con su criterio personal, a la vista de los resultados proporcionados por el ordenador.

El tipo de herramienta que aquí se considera incluye todos los procesos necesarios para evaluar completamente el diseño, pero dichas evaluaciones se emplean para alimentar a procesos de optimización; estos procesos de optimización integran algoritmos matemáticos convencionales junto a otros capaces de manejar el conocimiento (heurístico o no) de los expertos. De esta manera se intenta explorar sistemáticamente, pero no de forma exhaustiva, el espacio definido por el conjunto de variables independientes correspondientes a los grados de libertad del diseño. El resultado de esta exploración es una o varias soluciones que, cumpliendo las restricciones impuestas al diseño, son óptimas respecto a ciertos criterios de calidad.

Nota bibliográfica:
El origen del CAD se suele situar en la aparición de [Sutherland, 63].
Para distinguir las herramientas de diseño capaces de tomar decisiones de los sistemas tradicionales de CAD (que sólo evalúan) se propone el término ICAD (Intelligent CAD) en trabajos como [Dietterich & Ullman, 87], [Arbab, 89], [Waldron, 89]; en [Tomiyama & Ten Hagen, 87] se presenta el término IIICAD (Intelligent Integrated Interactive CAD); en [Kalay et al, 87] se distingue entre passive design models (para evaluación y representación) y active design models (sistemas inteligentes). En [Zhang, 89] se propone el término Automated CAD. En [Veinott, 72] se distingue entre Analysis Approach (evaluación) y Synthesis Approach (optimización).

Es habitual llamar "inteligentes" a las herramientas informáticas más sofisticadas, pero no parece nada clara la separación entre sistemas inteligentes y no inteligentes, ni en CAD ni en ningún otro área de programación. En esta tesis se considera que la resolución de cualquier problema (como pueda ser eliminar las líneas ocultas de un dibujo en 3-D) es en sí una actividad inteligente. También se considera aquí, y éste es un matiz muy importante, que el objetivo de una herramienta informática no es imitar la inteligencia humana, sino simularla y sustituirla en aquellas tareas en que esto resulte provechoso; lo importante es el "qué" hace la máquina y no el "cómo" lo hace.

La misión de la herramienta no es sustituir al ingeniero de diseño, sino hacerle más fácil razonar a nivel de sistemas; al tener ya integrados procesos de evaluación con algoritmos de optimización, el ingeniero puede estudiar cómo evolucionan las soluciones óptimas ante distintos requisitos y restricciones de diseño. Asimismo puede seleccionar finalmente la solución más adecuada, tomando tal vez en consideración factores o criterios de evaluación no incluidos en la herramienta.

En la literatura técnica de Ingeniería del Conocimiento aplicada al diseño por ordenador abunda este tipo de herramientas, aunque no todas incluyen procesos de optimización, contentándose la mayoría con cumplir ciertos requisitos de diseño.

Nota bibliográfica:
Algunas herramientas de diseño que se describen en la literatura técnica son: SYN en [De Kleer & Sussman, 80], CACE-III en [James et al, 85], EPOPEE en [Borisov & Naglis, 85], DWB y PROSCODE en [Tomiyama & Yoshikawa, 85], KAUS en [Ohsuga, 85], TOPOLOGY 1 en [Gero et al 85], HI-RISE en [Maher & Fenves, 85], MOLE en [Szalapag & Bijl, 85], EXCAP en [Darbyshire & Davies, 85], TIGRE en [Adiba & Nguyen, 85], MAPLE en [Bowen, 85], SIBEMOL y ADELE en [Cloarec et al, 85], FORLOG en [Dietterich & Ullman, 87], SIGHTPLAN en [Tommelein et al, 87], ALEX en [Kalay et al, 87], IDSIDES en [Kamal et al, 87], PROMPT en [Murthy & Addanki, 87], STRUPLE en [Maher & Zhao, 87], CIMS en [Iwata et al, 87], TRANSEPT en [Galiana & McGillis, 88], FBMS en [Shah & Rogers, 88], EDESYN en [Sukla, 88], M1 en [Gupta, 88], CATHEDRAL en [De Man et al, 90].

Algunas herramientas intentan proporcionar al usuario un entorno general para el diseño, mientras que otras son de aplicación muy específica; estas últimas responden a una visión más realista del problema del diseño y en la práctica ofrecen mejores resultados. En esta tesis no se

propone crear una herramienta informática de aplicación universal, sino una metodología general para desarrollar herramientas a la medida de las necesidades de cada usuario.

Los programas de optimización de diseño desarrollados según esta metodología gozarán de una estructura similar, utilizarán algoritmos parecidos y presentarán incluso un estilo común en su comunicación con el usuario. No obstante, su eficacia dependerá de las características de cada aplicación concreta, así como de los conocimientos y la creatividad del equipo encargado de su desarrollo. Este equipo estará formado tanto por expertos en diseño por ordenador como por expertos en el sistema que se desea diseñar.

La cooperación del equipo de ingenieros del conocimiento con el de ingenieros de diseño al que va destinada la herramienta es fundamental para el desarrollo y puesta a punto de la misma, así como para su refinamiento en versiones cada vez más completas. Además del aporte de conocimientos que representa, la participación del futuro usuario es imprescindible para que éste confíe en los resultados obtenidos y sepa interpretarlos. No se debe olvidar que la complejidad de los sistemas diseñados puede hacer que el tiempo de respuesta sea del orden de horas en un ordenador personal, por lo que la herramienta resultará poco atractiva para un usuario que no conozca en cierta medida su funcionamiento.

Un aspecto atractivo del desarrollo de un proyecto de optimización de diseño por ordenador es la necesidad de formalizar e implantar procesos utilizados por los ingenieros de diseño, lo que les obliga a reflexionar sobre sus propios conocimientos y a comprobar posteriormente su validez. Este proceso facilita enormemente la transmisión de conocimientos a otros ingenieros, reservando las tareas realmente creativas (difícilmente formalizables) a la intuición e imaginación personales.

La aplicación del tipo de herramientas que aquí se describe a la optimización del diseño de un sistema no siempre será recomendable. Para que el tiempo y el esfuerzo invertidos sean rentables el problema debe reunir ciertas características:

a) Las especificaciones del diseño (restricciones, criterios de optimización, etc) deben variar sensiblemente de un caso de estudio a otro, de forma que no es trivial el estimar un diseño óptimo a partir de otros ya conocidos.

b) El sistema a diseñar debe ser modelable matemáticamente en un plazo de tiempo razonable, en relación con la aplicación concreta. Es preferible además que el modelado

pueda realizarse a varios niveles distintos de detalle, para poder resolver la optimización del diseño por aproximaciones sucesivas.

c) El usuario, que habitualmente será un ingeniero de diseño, debe estar capacitado para comprender los principios de funcionamiento de la herramienta, de tal manera que pueda colaborar en su desarrollo y puesta a punto durante la etapa de pruebas.

Una característica muy importante de estas herramientas es que deben estar "vivas"; esto quiere decir que sus bases de datos puedan crecer, que los subprocesos que las componen puedan renovarse y que puedan incorporar nuevas prestaciones. La implantación informática debe ser modular y flexible, estando acompañada de una documentación exhaustiva y precisa.

En cuanto al alcance del contenido de esta tesis, se puede concretar a partir del análisis de su título: "El Problema General de la Optimización de Diseño por Ordenador: Aplicación de Técnicas de Ingeniería del Conocimiento."

Ya se ha especificado lo que aquí se entiende por "optimización de diseño por ordenador", un concepto más ambicioso que el de diseño asistido por ordenador (CAD). Una revisión de la literatura técnica relevante muestra, por el número y la variedad de trabajos publicados en los últimos años, que es un área de reciente desarrollo y alto interés en la que se está invirtiendo un gran esfuerzo en todo el mundo.

En esta tesis se considera que la Ingeniería del Conocimiento es simplemente la disciplina que permite desarrollar sistemas automáticos de resolución de problemas. Según esto cualquier problema resuelto por ordenador sería una aplicación de Ingeniería del Conocimiento. No obstante, la "aplicación de técnicas de Ingeniería del Conocimiento" se incluye en el título de la tesis para concretar el contenido de la misma en dos sentidos distintos:

a) El primero hace referencia a la visión global aquí propuesta sobre qué es la Ingeniería del Conocimiento y cómo se desarrolla un proyecto en esta disciplina. Conviene anticipar aquí que, según este trabajo, un proyecto así consta de tres fases: modelado, adquisición del conocimiento e implantación. El modelado consiste básicamente en la descomposición de un proceso de información en un conjunto de subprocesos relacionados entre sí, cada uno de los cuales deberá ser desarrollado a su vez como un proyecto de Ingeniería del Conocimiento.

b) El segundo corresponde al conjunto de técnicas específicas tradicionalmente englobadas dentro de la Inteligencia Artificial o Ingeniería del Conocimiento. El modelado de la Optimización de Diseño aquí propuesto plantea diversos subproblemas, alguno de los cuales puede abordarse siguiendo algunas de las técnicas ya conocidas.

En cuanto a tratarse de un "problema general", la formulación matemática y la estructura global aquí propuestas pretenden ser independientes de la aplicación, aunque algunos casos sean más apropiados que otros a la hora de la implantación; parece además bastante claro que el campo de aplicación se puede extender sin mucha dificultad a otros problemas complejos de optimización, como por ejemplo la planificación.

En la literatura existente sobre optimización de diseño pueden identificarse dos grandes corrientes: la primera nace de las técnicas convencionales de optimización matemática y cuenta con un conocimiento muy limitado del entorno de la Inteligencia Artificial; la segunda trata de aplicar las técnicas clásicas de la Inteligencia Artificial al campo de la optimización de diseño, tarea que no siempre tiene éxito porque estas técnicas son muy específicas del campo de aplicación original.

En ambas corrientes se encuentran trabajos interesantes pero en general de éxito modesto, debido en muchos casos a la falta de un modelo suficientemente abstracto y bien formalizado de lo que se entiende por optimizar un diseño. El enfoque más prometedor parece ser una síntesis de ambas corrientes: partir de un modelo matemático completo y riguroso para establecer el problema de diseño, aplicando luego las técnicas sofisticadas propias del entorno de la Inteligencia Artificial a la resolución práctica de dicho problema, teniendo en cuenta el conocimiento específico que pueden aportar los expertos.

Nota bibliográfica:
En [Allen, 90] estas dos corrientes se describen como "procedural and algorithmic approach" y "declarative (expert systems) approach"; en [Grossman, 90] se distingue entre "mathematical optimization approach" y "heuristic and knowledge engineering approach." En [Borrel, 87] se separa la "programación matemática" de la "programación dinámica" y de la "teoría de juegos."
En la corriente de optimización de diseño básicamente matemática se pueden incluir trabajos como [Veinott, 72], [Pérez Arriaga, 78], [Singh et al, 83], [Borisov & Naglis, 85], [Cuadra et al, 85], [García et al, 87], [Agogino & Almgren, 87], [Kamal et al, 87], [Appelbaum et al, 87], [Krishnan et al, 88], [Jazdzynski, 89], [Fei et al, 89], [Kermanshahi et al, 90], [Birewar, 90], [Grossman, 90], [Kuh & Ohtsuki, 90], [De Man et al, 90], [Maly, 90].
En la corriente de aplicación de la Inteligencia Artificial al diseño (sólo algunos trabajos tratan realmente la optimización) se pueden incluir [Tomiyama & Yoshikawa, 85], [Ohsuga, 85], [Gero et al, 85], [Maher & Fenves, 85], [Szalapag & Bijl, 85], [Mac an Airchinnig, 85], [Barbuceanu, 85], [Brown & Chandraserakan, 85], [Adiba & Nguyen, 85], [Bowen, 85], [Cloarec et al, 85], [Green & Brown, 87], [Han et al, 87], [Tommelein et al, 87], [Kalay et al, 87], [Maher & Zhao, 87], [Iwata et al, 87], [Yuzu et al, 87], [Coyne et al, 87], [Fenves & Baker, 87], [Shah & Rogers, 88], [Shukla, 88], [Waldron, 89], [Zhang, 89], [Woodbury & Oppenheim, 89].
Algunos trabajos que abordan el problema de la optimización y/o del diseño tratando de integrar los métodos matemáticos convencionales con técnicas de Ingeniería del Conocimiento son [Papalambros & Wilde, 79], [De Kleer & Sussman, 80], [James et al, 85], [Elías, 85], [Rehak et al, 85], [Kolb, 86a], [Kolb, 86b], [IIT, 87],

[Dietterich & Ullman, 87], [Ishii & Barkan, 87], [Struss, 87], [Kramer, 87], [Navinchandra & Marks, 87], [Galiana & McGillis, 88], [Brayton et al, 90], [McFarland et al, 90], [Allen, 90], [Cuadra & León, 90], [Cuadra et al, 90].

Un criterio más claro para clasificar los trabajos sobre diseño por ordenador puede ser el tipo de modelo escogido para caracterizar el proceso de diseño. Fundamentalmente hay dos tipos de modelos: el matemático y el lingüístico. En un modelo lingüístico el proceso de diseño se concibe como la construcción de un discurso significativo en un lenguaje específico. Dicho lenguaje cuenta con su propio léxico (variables, funciones, subsistemas, etc.), está sujeto a unas reglas de sintaxis (restricciones, ligaduras) y se interpreta según una semántica propia (evaluación); la sintaxis y la semántica se enriquecen por lo general con funciones matemáticas auxiliares que cubren los aspectos numéricos del proceso de diseño.

Nota bibliográfica:
El conflicto entre modelos lingüísticos y matemáticos se plantea explícitamente en [Waldron, 89]:

> "Should one study the design process in a linguistic framework or a mathematical one? The former allows a descriptive representation of the process variables as they exist, whereas in the latter the mathematics may not as yet be available or developed."

También en [Waldron, 89] se distinguen tres niveles del conocimiento relativo al diseño: el nivel semántico, en el que se establece el contexto del proceso global de diseño; el nivel sintáctico, o evaluación de la lógica y los criterios seguidos; y el nivel de forma, predominantemente numérico.
En [Fenves & Baker, 87] el proceso de diseño se describe como la combinación dinámica de objetos mediante operadores (generators) y reglas de formación que intentan mantener factibles las soluciones; ocasionalmente se aplican evaluadores (critics) que comprueban la factibilidad global del conjunto.
Algunos trabajos que usan modelos lingüísticos en el proceso de diseño son [Tomiyama & Yoshikawa, 85], [Ohsuga, 85], [Gero et al, 85], [Szalapag & Bijl, 85], [Mac an Airchinnig, 85], [Barbuceanu, 85], [Adiba & Nguyen, 85], [Han et al, 87], [Struss, 87], [Tomiyama & Ten Hagen, 87], [Tommelein et al, 87], [Kamal et al, 87], [Maher & Zhao, 87], [Iwata et al, 87], [Coyne et al, 87], [Fenves & Baker, 87], [Waldron, 89].
Un campo de aplicación en el que se utilizan ambos modelos (matemático y lingüístico) de forma natural es el diseño de circuitos lógicos (VLSI, Very Large Scale of Integration, ASIC, Application Specific Integrated Circuits). Esto es debido a que en estas aplicaciones hay que obtener la descripción detallada de un circuito físicamente fabricable a partir de un algoritmo descrito en un lenguaje de programación, por lo que los parámetros lingüísticos se deben transformar paso a paso en variables matemáticas.

En esta tesis se propone un modelo matemático para el proceso de optimización de diseño, junto a un enfoque integral y sistemático que permite abordar de forma eficaz un problema tan complejo. De ningún modo se pretende afirmar que dicho enfoque sea el único ni el mejor de los posibles; lo que sí se afirma es su coherencia y eficacia, pues tiene su origen en un método de trabajo que se ha seguido con distintas variantes durante más de diez años en proyectos reales, trabajando tanto individualmente como en equipo.

La reflexión sobre problemas concretos y sobre las soluciones adoptadas ha permitido abstraer y generalizar, llegando a un sistema coherente que abarca desde una definición de la ingeniería del conocimiento hasta detalles de implantación y métodos de representación específicos. Este sistema se expone a lo largo de las distintas secciones de la tesis según el esquema que se presenta a continuación:

En el Capítulo 2 se expone lo que en la tesis se considera que es la Ingeniería del Conocimiento, y cómo se puede aplicar esta disciplina al desarrollo de sistemas automáticos de resolución de problemas. Se describen las etapas de adquisición del conocimiento, modelado e implantación informática, y se propone para todas ellas el uso de lenguajes gráficos de representación del conocimiento.

En el Capítulo 3 se estudia el problema general de la optimización de diseño, se formaliza y modela dicho problema y se describe la idea central de la tesis: la Optimización Estructurada Multiatributo. A lo largo de los siguientes capítulos se tratarán con detalle los distintos aspectos del modelo y del enfoque propuestos.

En el Capítulo 4 se trata con detalle cómo puede llevarse a cabo una fase de diseño, es decir, una búsqueda sistemática capaz de hallar óptimos locales y globales en regiones de búsqueda complejas (con distintos tipos de variables y en espacios de búsqueda de dimensión variable) por medio de la descomposición del problema en una estructura de optimizaciones parciales.

El Capítulo 5 está dedicado al proceso de resolución de cada optimización parcial. Se estudian las estrategias de búsqueda que se acomodan a los distintos tipos de variables independientes: continuas, discretas y muy discretas, según los criterios de "tamaño" y "comportamiento local". En especial se describen las búsquedas locales y las búsquedas en árbol.

En el Capítulo 6 se trata la forma de abordar las optimizaciones con dos o más atributos y en especial con atributos mutuamente conflictivos, casos muy frecuentes en la práctica de la optimización de diseño. Se presentan los conceptos de hipersuperficie óptima, selección multiatributo y búsqueda multiatributo.

El Capítulo 7 está dedicado a la creación automática de funciones rápidas capaces de sustituir a procesos de cálculo lentos. Es un tema totalmente original de esta Tesis que ha surgido ante la necesidad real de incorporar procesos de cálculo lentos a programas de diseño con una estructura fuertemente iterativa. Se presentan en este capítulo métodos de interpolación en espacios de n dimensiones que se desarrollaron expresamente para solucionar problemas surgidos en proyectos reales. También se presenta una herramienta informática capaz de crear funciones rápidas que simulan los resultados de las funciones lentas originales por medio de interpolaciones lineales.

En el Capítulo 8 se exponen las conclusiones de la tesis y se señalan las aportaciones originales de la misma. También se comentan los aspectos del problema que quedan abiertos a futuros desarrollos y aplicaciones.

En el Apéndice A se ofrece una descripción del sistema gráfico de desarrollo y especificación de algoritmos ANA, utilizado a lo largo de la tesis y en algunos ejemplos reales de aplicación. Entre los resultados de la tesis debe incluirse la creación de una herramienta informática comercial capaz de generar interactivamente los diagramas ANA.

En el Apéndice B se describen brevemente los proyectos de optimización de diseño de los que ha surgido la metodología general que se expone a lo largo de la tesis.

Capítulo 2

La Ingeniería del Conocimiento

El mundo de la Ingeniería del Conocimiento (o Inteligencia Artificial, Sistemas Expertos, etc.) despierta en el entorno científico y técnico grandes entusiasmos y grandes recelos. Unos y otros parecen estar plenamente justificados debido a la desigual calidad de los proyectos y publicaciones englobados dentro de esta disciplina.

En particular, la comunidad de ingenieros se muestra especialmente desconfiada ante este tipo de trabajos, debido fundamentalmente a que está integrada por profesionales eminentemente prácticos a la hora de estimar los recursos invertidos en un proyecto. Uno de los objetivos de esta Tesis es contribuir a que esta desconfianza vaya desapareciendo paulatinamente, probando desde el propio entorno de la Ingeniería la viabilidad de proyectos de Ingeniería del Conocimiento y el interés de los resultados obtenidos.

Sin embargo, el tema central de la Tesis es la optimización de diseño, y no se pretende en ningún modo recopilar y evaluar la gran cantidad de publicaciones que existen sobre Inteligencia Artificial en general.

A lo largo de este capítulo se exponen las ideas generales que enmarcan todo el contenido de la Tesis. Estas ideas son inspiradoras de una línea de trabajo que ha resultado ser eficaz, coherente y relativamente fácil de aplicar a proyectos y entornos de trabajo muy diversos. Se describirá la nomenclatura utilizada, la metodología seguida y los razonamientos que hacen del conjunto un todo consistente.

En el apartado 2.1 se propone una definición rigurosa de la Ingeniería del Conocimiento y de su campo de actividad; en el apartado 2.2 se realiza una breve revisión del lenguaje y los

conceptos generales relacionados con la Lógica y la Ingeniería del Conocimiento; en el apartado 2.3 se propone un método sistemático para desarrollar proyectos de Ingeniería del Conocimiento; en el 2.4 se presenta la metodología ANA de representación gráfica de modelos, algoritmos y programas; en el apartado 2.5 se discute la conveniencia de usar lenguajes de programación de bajo nivel (y en particular el lenguaje C); finalmente, el apartado 2.6 está dedicado a comentar aspectos relacionados con este capítulo en los proyectos reales de diseño por ordenador que han dado lugar a esta tesis.

2.1 - Definición de Ingeniería del Conocimiento.

Es deseable que un trabajo científico pueda enmarcarse con precisión en uno o varios campos de actividad definidos. Por desgracia la Ingeniería del Conocimiento es una actividad poco definida y que se encuentra inmersa en una gran confusión de vocabulario. Esto es debido no sólo a su novedad sino a su carácter interdisciplinario, pues la desarrollan tanto filósofos como matemáticos, ingenieros o programadores.

En este apartado se propone una definición sencilla, rigurosa y que puede seguir siendo válida (al menos su contenido) independientemente del futuro desarrollo de la disciplina. Las definiciones que aparecen en la literatura técnica revisada son por lo general poco rigurosas, vagas y dependientes del estado del arte.

Nota bibliográfica:

En [Barr & Feigenbaum, 81] se propone la siguiente definición para Inteligencia Artificial:

> "Artificial Intelligence is the part of computer science concerned with designing intelligent computer systems, that is, systems that exhibit the characteristics we associate with intelligence in human behaviour - understanding language, learning, reasoning, solving problems and so on."

También en [Barr & Feigenbaum, 81] se define Ingeniería del Conocimiento como la tarea de desarrollar Sistemas Expertos, y a su vez estos últimos son definidos como:

> "Expert Systems can be viewed as intermediaries between human experts, who interact with the systems in 'knowledge adquisition' mode, and human users who interact with the systems in 'consultation' mode."

En [Rich, 83] se propone la siguiente definición para Inteligencia Artificial:

> "Artificial Intelligence is the study of how to make computers do things at which, at the moment, people are better."

También se habla en [Rich, 83] de los Sistemas Expertos y la Ingeniería del Conocimiento:

> "There is a whole array of interesting tasks...that require a great deal of specialized knowledge... These tasks can only be performed by experts who have accumulated the required knowledge... Such programs are called expert systems and the construction of them is referred to as knowledge engineering."

En [Winston, 84] se da esta definición:

> "Artificial Intelligence is the study of ideas that enable computers to be intelligent."

En [Charniak & McDermott, 85] aparecen las definiciones siguientes:

> "Artificial Intelligence is the study of mental faculties through the use of computational models."
>
> "An expert system is a rule-based Artificial Intelligence application program for doing a task which requires expertise."

En [Ohsuga, 85] se proponen las siguientes definiciones:

> "Artificial Intelligence is the area in which one analyzes human intelligence and tries to give computers the intelligent faculties."
>
> "Knowledge Engineering is a practical research area in which one intends to design practical intelligent computer systems using known methods and means."

En [Gero et al, 85] se define indirectamente la Ingeniería del Conocimiento como el arte de trasladar el conocimiento de los expertos a una base de datos, pues se especifica que una forma de adquisición del conocimiento es:

> "...from the expert to the knowledge base via a knowledge engineer."

En [Maher & Zhao, 87] se define un sistema experto como:

> "...an appropiate tool for assisting humans in solving the problems which can not be defined mathematically but are solved daily by human experts using both domain knowledge learned from books and experiences gained from practice."

El objeto de este apartado no es discutir en detalle las definiciones existentes. Simplemente se pretende respaldar la coherencia de la nomenclatura aquí utilizada con la línea de trabajo y pensamiento propuesta para facilitar la comprensión del conjunto. Las definiciones auxiliares empleadas en los siguientes párrafos han sido extraídas de una enciclopedia de uso extendido [Larousse, 67] para garantizar la objetividad y general aceptación de las mismas.

La definición de Ingeniería del Conocimiento que aquí se propone está basada en la definición de **Ingeniería** que aparece en [Larousse, 67] :

> "Ingeniería es el arte de aplicar los conocimientos científicos a la invención, perfeccionamiento y utilización de la técnica industrial en todas sus dimensiones."

Se supone que la dimensión de la técnica industrial a la que se aplica la Ingeniería del Conocimiento es la **resolución automática de problemas** porque "resolver un problema" es un concepto suficientemente amplio para abarcar a cualquier proceso de pensamiento que persiga fines prácticos. Por ejemplo, en [Stebbing, 65] se afirma:

> "El pensamiento reflexivo consiste esencialmente en el intento de resolver un problema...nuestros pensamientos están dirigidos hacia un fin: la solución del problema que nos puso a pensar."

La naturaleza de los conocimientos científicos mencionados en la definición de Ingeniería depende de cada especialidad concreta, aunque todas tienen en común la Física y una fuerte base matemática; la resolución de los problemas a los que se enfrenta la Ingeniería exige la

adaptación de las leyes de la Física a distintos modelos matemáticos según la precisión que requiera cada aplicación.

La Ingeniería del Conocimiento debe estar ligada a una disciplina que estudie el Conocimiento en sí, esto es, la Filosofía. Pero en el ámbito de la Filosofía el problema del Conocimiento se estudia a tres niveles distintos, según [Larousse, 67]:

- El origen del Conocimiento, estudiado por la Psicología.
- Los criterios del Conocimiento, estudiados por la Lógica.
- La naturaleza del Conocimiento, estudiado por la Metafísica.

Según [Wartofsky, 68] las tres grandes áreas de la Filosofía son otras, aunque los objetivos de la Metafísica y de la Lógica sigan siendo básicamente los mismos:

- La Metafísica estudia qué existe y cuál es la naturaleza de lo que existe.
- La Epistemología estudia cómo podemos conocer las cosas que existen y cómo justificamos nuestras pretensiones de conocimiento.
- La Lógica estudia cómo se relacionan los conceptos entre sí, qué es una inferencia válida o razonamiento correcto y qué es la verdad.

Ahora bien, todas las Ingenierías tradicionales tienen en común el modelado del comportamiento del Universo a partir de las leyes de la Física (aparte de la base matemática). Sin embargo no es cometido del ingeniero estudiar el origen ni la naturaleza del Universo, sino aplicar los modelos de su funcionamiento más adecuados. Del mismo modo en la Ingeniería del Conocimiento sólo interesa (con fines prácticos) estudiar la **Lógica**, es decir, cómo modelar el funcionamiento del Conocimiento y cómo pueden utilizarse esos modelos en provecho de la colectividad.

La definición que aquí se propone es, por lo tanto, la siguiente:

> "La **Ingeniería del Conocimiento** es el arte de aplicar los conocimientos de la Lógica a la invención, perfeccionamiento y utilización de sistemas automáticos de resolución de problemas."

Se observará que el término "conocimiento" aparece tanto en la definición como en lo definido, lo que a primera vista parece descalificar la propia definición. Sin embargo dicha aparición es en el fondo inevitable, pues es reflejo del proceso intrínsecamente recursivo de que un ser humano utilice su pensamiento para reflexionar sobre el pensamiento mismo, e incluso

para producir otros entes capaces de emular y sustituir con éxito algunas actividades propias de su pensamiento (como son las herramientas informáticas).

El mismo problema aparece en la expresión "conocimientos de la Lógica", y sin embargo no impide su comprensión. En efecto, según [Larousse, 67] la definición de **Lógica** es:

> "Disciplina que estudia la estructura, fundamento y usos de las expresiones del conocimiento humano."

Según la definición que aquí se propone la realización de cualquier programa, por sencillo que sea, se considera un proyecto de Ingeniería del Conocimiento. Efectivamente, el salto entre programas sencillos y los grandes proyectos de "Inteligencia Artificial" no es cualitativo sino cuantitativo, al igual que, por ejemplo, el salto entre diseñar una grapadora y diseñar una nave espacial.

Nota bibliográfica:
Está muy extendida la idea de separar la "Inteligencia Artificial" de la programación convencional, aunque esto no parece muy convincente porque tal distinción se suele basar simplemente en criterios de complejidad y de arquitectura de programación. Por ejemplo, en [Darbyshire & Davies, 85] se distingue entre "expert systems" y "algorithms"; en [Hatvany, 85] se distingue entre "problem-solving systems" y "algorithmic programs"; en [Han et al, 87] se opone "knowledge-based programming" a "ordinary programming"; en [Tommelein et al, 87] se enfrentan "algorithmic models" y "procedural methods" a "Artificial Intelligence approach"; en [De Man et al, 90] se distingue entre "rule-based techniques" y "algorithm-based techniques"; en [Allen, 90] se opone "procedural or algorithmic approach" a "declarative (expert systems) approach".
Es evidente sin embargo que cualquier sistema informático resuelve algún problema (problem-solving) por medio de un algoritmo (algorithm). También es evidente que un algoritmo incluye procedimientos (procedures) y que está basado en el conocimiento (knowledge-based), aunque dicho conocimiento pueda estar explícitamente recogido en una base de datos (modelo declarativo: así suelen construirse los sistemas expertos clásicos) o implícitamente incluido en las funciones y en la estructura de control del algoritmo (modelo procedural).

2.2 - Lógica e Ingeniería del Conocimiento.

La Ingeniería del Conocimiento es la aplicación práctica de los métodos y principios de la Lógica, y por tanto cabría esperar que los textos de Ingeniería del Conocimiento y los de Lógica tuvieran mucho en común. Sin embargo, los textos de Lógica van dirigidos a estudiantes de Filosofía, por lo que son rigurosos pero poco prácticos, mientras que los textos de Ingeniería del Conocimiento son prácticos pero poco rigurosos, pues tienen su origen y están orientados a la resolución Informática de problemas concretos.

Un trabajo en el que Lógica, Ingeniería del Conocimiento y Diseño se tratan simultáneamente de una forma coherente y global es [Coyne et al, 87]. En este apartado se utilizan las fuentes de Lógica y de Ingeniería del Conocimiento por separado, intentando extraer unos criterios y un vocabulario comunes a ambas disciplinas. Se intenta dar así una visión general del tema que, naturalmente, será muy breve y simplificada, por no ser el asunto central de esta tesis.

2.2.1 - Lógica.

Lo primero que cabe destacar es la existencia de varios tipos de Lógica o de varias lógicas; se advierte sin embargo que las clasificaciones que se ofrecen a continuación no son excluyentes: algunos tipos de lógica pueden combinarse entre sí en ciertos problemas, aparte de que sea discutible si algunos tipos de lógica no son simplemente una variedad de otros.

En [Deaño, 83] se distingue para empezar entre lógica **clásica** y lógicas **no-clásicas**. La lógica clásica se identifica con la llamada **Lógica de Predicados**, que a su vez puede ser de primer orden o de orden superior:

- En la de primer orden sólo se admite aplicar cuantificadores a las variables.

- En las de orden superior se puede aplicar cuantificadores tanto a las variables como a las propiedades.

En [Deaño, 83] se agrupan las llamadas lógicas **no-clásicas** (o también lógicas desviadas) en dos grandes familias: las lógicas polivalentes y las lógicas modales.

Las **lógicas polivalentes** son aquellas en las que se admiten más de dos valores de certeza para los asertos (en la clásica sólo se admiten dos, verdadero y falso). Las dos lógicas más importantes de este grupo son la Lógica Difusa y la Lógica Presuposicional:

- La **Lógica Difusa** [Zadeh, 65] admite infinitos grados de certeza entre la verdad absoluta y la falsedad absoluta, permitiendo que los conjuntos definidos por las propiedades de los individuos tengan fronteras "borrosas". Está relacionada con los modelos llamados estocásticos o probabilistas, mientras que las lógicas bivalentes se asocian a modelos deterministas.

- La **Lógica Presuposicional** ([Van Fraasen, 69], citado en [Deaño, 83]) admite que el conocimiento del universo del discurso no sea completo, permitiendo por tanto el razonamiento "por defecto" y el estado de "creencia" para asertos probables pero no confirmados; dicho estado se puede revisar si aparece información contradictoria, por lo que esta lógica es **no-monotónica**.

Las **lógicas modales** son aquellas en las que se tratan los matices o modalidades (no los valores intermedios) de la verdad y la falsedad, en contra de la lógica clásica que se limita a afirmar o negar las proposiciones. En [Deaño, 83] se citan como lógicas modales las modalidades aléticas, las epistémicas, las existenciales, las deónticas, la lógica de los mandatos, la de la preferencia, la erotérica y la lógica cronológica:

- En las **modalidades aléticas** (o de verdad) los operadores modales son "necesario", "posible", "imposible" y "contingente", y tratan las relaciones de inferencia entre enunciados afectados por dichos operadores ([Carnap, 47], citado en [Deaño, 83]).

- En las **modalidades epistémicas** los operadores son "verificado", "no decidido" y "refutado"; en las **modalidades existenciales** los operadores son "universal", "existente" y "vacío".

- Las **modalidades deónticas** se ocupan de las relaciones de inferencia entre normas, es decir, entre proposiciones prescriptivas. Sus operadores son: "obligatorio", "permitido" y "prohibido" ([Wright, 51], citado en [Deaño, 83]).

- La **lógica de los mandatos** trata los enunciados imperativos ([Rescher, 66], citado en [Deaño, 83]); la **lógica de la preferencia y la elección** se ocupa de las relaciones de inferencia entre enunciados estimativos ([Wright, 63], citado en [Deaño, 83]); la **lógica erotérica** se ocupa de los enunciados interrogativos, o también de las relaciones entre preguntas y respuestas ([Harrah, 63], citado en [Deaño, 83]).

- La **lógica cronológica** o lógica del tiempo surge tanto del reconocimiento de esquemas de inferencia específicamente temporales como de la importancia del factor tiempo (tiempo de los verbos) en los enunciados de un razonamiento ([Gardies, 75], citado en [Deaño, 83]).

Nota bibliográfica:
La existencia de esta variedad de lógicas es frecuentemente ignorada, debido a que en la práctica la mayoría de los proyectos tradicionalmente englobados en la Ingeniería del Conocimiento están basados en la Lógica de

Predicados de primer orden. Por ejemplo, en [Gero et al, 85] se limita la Ingeniería del Conocimiento a este tipo de lógica, pues se define "logic programming" como:

"...the process of encoding first order predicate logic into computer programs."

En [Dietterich & Ullman, 87], [Ishii & Barkan, 87] se asocia el término "logic programming" a sistemas basados en "facts" y "rules".

Además del tipo de lógica empleado para plantear un problema se pueden clasificar los tipos de procesos de pensamiento (o razonamientos) empleados para resolverlo. En Lógica se distingue entre el **razonamiento deductivo** y el **razonamiento inductivo**. Según [Stebbing, 65]:

> "Cuando la conclusión es implicada por las premisas, el razonamiento es deductivo; cuando las premisas no son suficientes para implicar la conclusión, pero no obstante tienen cierto peso como evidencia en favor de ella, se dice que el razonamiento es inductivo."

Según esto, las lógicas polivalentes en general son sólo útiles para realizar razonamientos **inductivos**, y así sucede en cualquier procedimiento basado en la acumulación de evidencia. Debe tenerse en cuenta que, bien por usar modelos incompletos de los problemas o por usar información incompleta, no siempre se puede afirmar la total certeza o falsedad de un aserto. Según [Stebbing, 65] todo esto se engloba dentro de modelos **probabilistas**:

> "Las premisas pueden proporcionar evidencia en apoyo de la conclusión sin proporcionar lógicamente evidencia concluyente; en este caso se dice que la relación es una relación de probabilidad."

Nota bibliográfica:

En [Coyne et al, 87] se distinguen tres formas de razonamiento: deducción, inducción y abducción; precisamente se propone el diseño como un proceso de razonamiento abductivo. Estas formas se describen (según el orden citado) como sigue:

> "In making deductions we employ rules to chain through related propositions to establish the consistency of new statements with given statements."
> "...induction...the process by which logical rules are derived."
> "...abduction...the derivation of statements about the world given logical rules and some logical consequences."

Todo lo expuesto se refiere al razonamiento o inferencia como método de hallar la solución de un problema. Sin embargo hay problemas cuya solución se obtiene por **analogía** (reconocimiento de una estructura común). Tal es el caso del reconocimiento de símbolos y patrones, tarea en la que el ser humano es especialmente hábil y que es fundamental en gran número de las actividades llamadas "inteligentes". Entre éstas se pueden destacar el aprendizaje, la visión y la comprensión de lenguajes.

2.2.2 - Ingeniería del Conocimiento.

La aspiración de la llamada **tradición algorítmica** en Lógica consiste en reducir el razonamiento al cálculo. Los principales representantes de esta tradición, según [Sacristán, 64], son Ramón Lull (1235-1315) y G. Leibniz (1646-1716). El éxito de este enfoque está limitado por la capacidad de formalizar y modelar los problemas con corrección y por la capacidad de cálculo en sí. Tanto los problemas como sus métodos de resolución deben poder primero expresarse en lenguaje matemático y luego deben poder resolverse en un tiempo razonable.

La Ingeniería del Conocimiento se puede considerar una revitalización espectacular de esta tradición; esto ha sido posible por la aparición del ordenador y el inmenso avance que supone su capacidad de cálculo. En cuanto a avances en el terreno de la formalización de los problemas, quizá el más significativo sea la noción de incertidumbre y los métodos para tratarla en todas sus facetas.

Realizar un proyecto de Ingeniería del Conocimiento implica transformar un problema y un método de resolución en una herramienta informática, o sea, en un programa de ordenador. Dado lo difícil que sería clasificar exhaustiva y objetivamente todos los problemas y métodos de resolución posibles, es preferible comenzar por analizar el producto final del proceso, es decir, los programas de ordenador.

Un ordenador consiste básicamente en una base de datos (memorias) gestionada por un conjunto de operadores (procesadores). Estos últimos también se encargan de gestionar las comunicaciones de la máquina con el exterior, pero lo que importa destacar ahora es la tarea interna de proceso de información. Los elementos de esta tarea son los datos y los procesos: los **datos** se almacenan en la base de datos, mientras que los **procesos** se encargan de cambiar el estado de dicha base de datos.

La secuencia de actuación de los procesos depende a su vez del estado de la base de datos, con lo cual en teoría se puede alcanzar cualquier nivel de complejidad y de abstracción. Esto permite plantear la posible creatividad e innovación en las soluciones ofrecidas por herramientas informáticas, como se verá en el apartado 2.5.

Es inmediato establecer el paralelismo entre los datos y procesos de la máquina y las **variables** y **funciones** matemáticas. También es inmediato presentar el concepto de algoritmo; en [Larousse, 67] se define **algoritmo** como:

"Conjunto de símbolos y procedimientos de los cálculos matemáticos."

Según este razonamiento toda aplicación informática es simplemente la implantación de un algoritmo; por ello no parecen correctas las ya mencionadas distinciones entre "Inteligencia Artificial" y "programación algorítmica" que se encuentran, por ejemplo, en [Darbyshire & Davies, 85], [Hatvany, 85] y [Tommelein et al, 87].

En el lenguaje clásico de Ingeniería del Conocimiento se usan los términos de **conocimiento estático** y **conocimiento dinámico**; en este trabajo se emplean los términos matemáticos de variables y funciones. En el apartado siguiente se expondrá cómo el desarrollo de un algoritmo consiste en la creación de dos estructuras relacionadas entre sí: la **estructura de variables** y la **estructura de funciones**. También se mostrará que ambas tienen una forma común, que es la de **red de árboles**.

Se ofrece a continuación un breve repaso a las técnicas de Ingeniería del Conocimiento más extendidas y a sus relaciones con el entorno de la Lógica. Para ello se han utilizado varias fuentes, entre las que se cuentan [Barr & Feigenbaum, 81], [Rich, 83], [Winston, 84] y [Charniak & McDermott, 85].

En el ámbito de la Ingeniería del Conocimiento se reconocen las dos corrientes filosóficas que han configurado el pensamiento y la ciencia de la Era Moderna: el Racionalismo y el Empirismo. En la primera se parte de la razón para deducir modelos precisos de la realidad, mientras que en la segunda se parte de la experiencia para inducir dichos modelos.

Según las definiciones citadas en el apartado anterior, los llamados "sistemas expertos" responden a un enfoque empirista de la resolución de problemas. La solución eficaz de un problema real exige un equilibrio entre ambas posturas, equilibrio impuesto por la naturaleza de cada problema concreto. Por ejemplo, las ciencias más experimentales (medicina, biología, etc.) son campos en los que aplicar sistemas expertos y enfoques probabilistas parece más adecuado que intentar aplicar modelos detallados y rigurosos (en [Maher & Zhao, 87] los sistemas expertos se consideran destinados a resolver problemas que no puedan definirse matemáticamente).

Se pueden identificar dos tareas fundamentales a las que se enfrenta la Ingeniería del Conocimiento al intentar automatizar la resolución de un problema, tareas que por razones de eficacia están estrechamente relacionadas:

- La **representación** formal y la **organización** del conocimiento (tanto estático como dinámico) relativo al problema.

- La **búsqueda** de soluciones utilizando unos recursos limitados, tanto de tiempo como de memoria.

En cuanto a la **búsqueda de soluciones** merecen destacarse dos tipos de enfoques: la búsqueda heurística y la planificación. Entendiendo la solución de un problema como una sucesión de **estados** entre un estado inicial y un estado final (solución) y una sucesión de **operadores** que permiten el paso de un estado a otro, estos enfoques se pueden describir así:

- La **búsqueda heurística** se basa en la evaluación de muchos estados alternativos y un control sencillo de la aplicación de los operadores. La evaluación se realiza por medio de un conjunto de parámetros, basados en la experiencia o en modelos simplificados del problema, llamados **heurísticos**. Los heurísticos son estimadores de la calidad de la solución que se espera obtener a partir de un estado intermedio dado, y sirven para evaluar cada estado provisional.

- La **planificación** se basa en un control sofisticado de la aplicación de los operadores y en la evaluación de menos estados. Es característico de este enfoque la descomposición estructurada del problema en subproblemas (subobjetivos), y se aplica frecuentemente a problemas en los que el tiempo es una variable crítica (por ejemplo el problema de diseño planteado y resuelto en [Birewar, 90]).

Los dos enfoques anteriores no son en absoluto incompatibles, sino en muchos casos complementarios. La unión de la evaluación de estados con heurísticos y de criterios complejos de control de los operadores puede producir algoritmos de búsqueda muy eficaces.

Nota bibliográfica:
En [Coyne et al, 87] se describe el proceso de solución de un problema de diseño como:

> "...changes in states under the transformations brought by various operators under a control strategy."

El término "heurístico" significa "relativo a la invención y al ingenio", y se suele utilizar como oposición a lo rutinario y a lo matemáticamente probado, con ciertas connotaciones de imprecisión.
Ejemplos de aplicación de búsqueda heurística son [Dietterich & Ullman, 87], [Navinchandra & Marks, 87].

Ejemplos de planificación son los sistemas con arquitectura de "pizarra" (blackboard), como [Hayes-Roth, 85], [Barbuceanu, 85], [Rehak et al, 85] y [Tommelein et al, 87], [Gupta, 88]. Otros ejemplos de planificación son [Kalay et al, 87], [Murthy & Addanki, 87], [Fenves & Baker, 87], [Birewar, 90].
Algunos sistemas que integran la búsqueda heurística y la planificación son [Yuzu et al, 87], [Cuadra & León, 90].

Como ya se ha comentado, la **analogía** es en sí un método de resolución de ciertos problemas. En la Ingeniería del Conocimiento se intenta implantar sistemas capaces de reconocer imágenes, lenguajes "naturales" (llamados así para distinguirlos de los lenguajes de ordenador) y situaciones por medio de la analogía. Como es fácil suponer, la analogía juega también un papel fundamental en los sistemas de aprendizaje automático.

Nota bibliográfica:
En [Carbonell, 85] se define el proceso de "analogical problem solving" como:

> "...consists of transferring knowledge from past problem solving episodes to new problems that share significant aspects with corresponding past experience and using the transferred knowledge to construct solutions to the new problems."

Algunos ejemplos concretos de aplicación de la analogía y el reconocimiento de patrones al diseño son [Bowen, 85], [Maher & Zhao, 87], [Zhang, 89].
En [Zhang, 89] se propone la analogía a distintos niveles de abstracción como forma de lograr un diseño automático "creativo"; concretamente se propone la analogía por "functional equivalence".
En [Kuh & Ohtsuki, 90] se cita un método de resolver la optimización del "routing" en circuitos VLSI basado en la analogía existente entre dicho problema y la minimización de pérdidas en una red eléctrica, aprovechándose así los algoritmos y herramientas ya existentes en este campo (como por ejemplo el tratamiento de matrices vacías).

Ya se ha comentado la tarea de búsqueda de soluciones. Ahora se describirán las formas más habituales e interesantes de abordar el problema de la **representación y organización del conocimiento**. Se han seleccionado las siguientes: las redes semánticas, los "marcos" (frames) y "guiones" (scripts), la lógica de predicados, el razonamiento cualitativo, diversas formas de razonamiento aproximado, y por último el razonamiento funcional o "procedural".

Las redes semánticas, "marcos" y "guiones" son organizaciones estructuradas de datos y procedimientos que facilitan ciertos tipos de razonamientos:

- Las **redes semánticas** (semantic networks) representan los entes de un problema (objetos, conceptos, procedimientos) como nodos de una red, estando los nodos unidos los unos a los otros por distintos tipos de relaciones ("ser parte de", "ser caso particular de", "ser equivalente a", etc.). Esta estructura facilita razonamientos asociativos (analogía) y se ha empleado mucho en programas educativos y de comprensión de lenguajes. Una visión general de las redes semánticas se ofrece en [Findler, 79].

- Los "**marcos**" (frames, [Minsky, 75]) y "**guiones**" (scripts, [Shank & Abelson, 77]) agrupan los datos y procedimientos relacionados con un determinado ente en una

estructura propia de la clase o tipo al que pertenece dicho ente. Esto permite aprovechar la herencia de propiedades a distintos niveles de abstracción y facilita los procesos de razonamiento presuposicional basados en el contexto. Estas estructuras están estrechamente relacionadas con un estilo particular de programación, la programación orientada al objeto (object oriented programming, [Cook, 86]).

Nota bibliográfica:
Como aplicaciones de estos tipos de organización del conocimiento al campo del diseño se pueden citar [Barbuceanu, 85], [Tommelein et al, 87], [Iwata et al, 87], [Galiana & McGillis, 88], [Shah & Rogers, 88], [Shukla, 88], [Zhang, 89], [Woodbury & Oppenheim, 89].

La **Lógica de Predicados** representa el conocimiento estático como **hechos** o enunciados, y el dinámico como **reglas** (operadores). Las reglas establecen la veracidad o falsedad de ciertos hechos a partir de la veracidad o falsedad de otros. Esta representación se utiliza para efectuar razonamientos de tipo **deductivo**, tanto en sentido directo (forward reasoning) como inverso (backward reasoning, o abducción según [Coyne et al, 87]), por medio de relaciones de implicación. Cuando es en sentido directo, a partir de las premisas se buscan conclusiones, mientras que en sentido inverso se supone una conclusión y se comprueba su certeza o falsedad comprobando las premisas.

Nota bibliográfica:
Como ejemplos de la lógica de predicados aplicada al diseño se pueden citar [Tomiyama & Yoshikawa, 85], [Ohsuga, 85], [Gero et al, 85], [Maher & Fenves, 85], [Mac an Airchinnig, 85], [Cholvy & Foisseau, 85], [Cloarec et al, 85], [Dietterich & Ullman, 87], [Tomiyama & Ten Hagen, 87], [Han et al, 87], [Ishii & Barkan, 87], [Green & Brown, 87], [Yuzu et al, 87], [Coyne et al, 87], [Shukla, 88], [Galiana & McGillis, 88], [Zhang, 89].

El **razonamiento cualitativo** permite sustituir modelos matemáticos complejos por sistemas de lógica de predicados. Para ello se discretiza cada variable a unos cuantos valores significativos (alto, medio, bajo,...) y se sustituyen las funciones complejas por un conjunto de reglas que relacionan los estados de las variables. En [Rainman, 86] se utiliza el término "order of magnitude reasoning" en vez de razonamiento cualitativo.

El **razonamiento aproximado** es un término general para englobar distintas aplicaciones de las lógicas polivalentes. En este bloque se pueden agrupar los modelos probabilistas, la lógica difusa y los sistemas de acumulación de evidencia.

El **razonamiento no-monotónico** ([Reiter, 80], [Etherington, 88]) está estrechamente ligado a la lógica presuposicional, y es el fundamento de los llamados "sistemas de conservación de la verdad" (TMS, Truth Maintenance Systems). En este tipo de enfoques se admite el razonamiento por omisión según suposiciones hechas a priori; estas suposiciones y

las conclusiones que de ellas se derivan se revisan cuando aparece información adicional, pudiendo modificarse dinámicamente el estado de certeza o falsedad de los hechos.

Nota bibliográfica:
Se pueden citar como ejemplos de razonamiento cualitativo aplicado a diseño [Green & Brown, 87], [Ishii & Barkan, 87], [Agogino & Almgren, 87].
Como aplicaciones del razonamiento aproximado al diseño se puede citar [Borisov & Naglis, 85], [Bowen, 85], [Maher & Zhao, 87], [Navinchandra & Marks, 87], [Maly, 90], [Mozumder & Strojwas, 90].
Un ejemplo de aplicación de razonamiento no-monotónico al diseño es [James et al, 85], donde se utiliza el término RMS (Reason Maintenance System).

Una herramienta informática se puede concebir como una estructura compleja de funciones interrelacionadas por intercambios y propagación de datos, siguiendo un modelo de cerebro humano. El problema fundamental que se presenta en este modelo es el control de la convergencia, estudiado bajo el término general de "**propagación de restricciones**" (constraint propagation). En el marco de este enfoque se pueden citar las redes neuronales y el razonamiento funcional:

- Las **redes neuronales** (neural networks, [Vemuri, 88]) o sistemas conexionistas están formadas por un gran número de funciones muy elementales conectadas entre sí por relaciones a las que se conceden distintos pesos relativos o importancias. Estos pesos relativos pueden autoajustarse según los resultados obtenidos en los razonamientos, permitiendo así un cierto nivel de aprendizaje automático. Su capacidad de aprendizaje y la posibilidad de construir circuitos integrados (hardware) con esta estructura hace que las redes neuronales sean muy interesantes de cara al futuro.

- El **razonamiento funcional** (functional reasoning, procedural reasoning) está basado en la definición de funciones más complejas que las utilizadas en las redes neuronales, pero en número más reducido. Es el fundamento teórico de estilos de programación en la práctica muy distintos, como puede ser la programación estructurada [Dahl et al, 72] y la programación funcional ([Bird & Wadler, 88], [Field & Harrison, 88]).

Nota bibliográfica:
El razonamiento funcional se utiliza ampliamente en el diseño de circuitos lógicos (hardware) a distintos niveles de detalle, desde "logic blocks" hasta "boolean networks". Se considera aquí que los algoritmos (y por tanto también el software) pueden diseñarse tratándolos como circuitos lógicos de muy alto nivel de abstracción.
El razonamiento funcional y la propagación de restricciones se han aplicado al diseño en [Freeman & Newell, 71] y más recientemente en [De Kleer & Sussman, 90], [Bowen, 85], [Rehak et al, 85], [Dietterich & Ullman, 87], [Struss, 87], [Kramer, 87], [Kamal et al, 87].

El modelo general de desarrollo e implantación de algoritmos propuesto en esta tesis encaja dentro de lo descrito como razonamiento funcional. Esto responde a la idea de que cualquier

técnica de Ingeniería del Conocimiento puede considerarse una forma de razonamiento funcional. Por ejemplo, las reglas propias de la Lógica de Predicados no son más que funciones entre las premisas (datos de entrada) y las conclusiones (datos de salida), y los hechos no son más que variables que pueden tomar dos o más (si se trata de lógicas polivalentes) valores distintos, que corresponden a su grado de certeza.

La interpretación de toda automatización de razonamientos como una forma de razonamiento funcional (modelo procedural) no se basa en un modelo del cerebro ni del conocimiento humano en general, sino en criterios más objetivos y prácticos: el de la naturaleza de los procesos realmente llevados a cabo por un ordenador (el ordenador tiene que resolver el problema con sus propios recursos) y en la posibilidad de una representación gráfica unívoca del conocimiento.

Nota bibliográfica:
La elección de la Lógica Funcional representa en el fondo una opción entre dos corrientes opuestas en la Ingeniería del Conocimiento: la declarativa y la procedural. Esta controversia se describe así en [Winograd, 75]:

> "...an Artificial Intelligence incarnation of the old philosophical distinction between 'knowing that' and 'knowing how'... Proceduralists assert that our knowledge is primarily 'knowing how'. The human information processor is a stored program device, with its knowledge of the world embedded in the programs... The declarativists, on the other hand, do not believe that knowledge of a subject is intimately bound with the procedures of its use. They see intelligence as resting on two bases: a quite general set of procedures for manipulating facts of all sorts, and a set of specific facts describing particular knowledge domains."

Es muy importante destacar lo que a continuación se afirma:

> "From a strictly formal point of view there is no difference between these positions... Declarativists and proceduralists differ in their approach to the duality between modularity and interaction, and their formalisms are a reflection of this viewpoint."

En este mismo trabajo se analizan con precisión las ventajas e inconvenientes de ambas posturas, pero es fundamental la aclaración que se hace sobre los criterios de valoración seguidos:

> "In this entire discussion, we could divide the question into two aspects: 'What kind of representation do people use?' and 'What kind of representation is best for machine intelligence?' I will not make this distinction..."

Es evidente que esta distinción desequilibra por sí sola la balanza a favor del modelo procedural, pues las máquinas son netamente procedurales (procesadores y memorias, funciones y variables). Teniendo en cuenta que no se considera la realidad de la máquina, las ventajas de ambas posturas que se enumeran en [Winograd, 75] son:
Por parte de la declarativa:

- Flexibilidad (no hay que especificar cómo y cuándo se hacen las cosas, sólo añadir hechos y reglas)
- Economía (los mismos hechos se pueden emplear para distintos razonamientos)
- Comprensibilidad y comunicabilidad (se parece más al lenguaje natural)
- Aprendizaje y accesibilidad (se pueden añadir hechos con facilidad)

Por parte de la procedural:

- Modelos procedurales (la naturaleza de muchos problemas es procedural)
- Posibilidad de abstracción (cada decisión se aplica en su contexto)
- La eficacia de aplicar procedimientos específicos a problemas locales (heurísticos)

Además de la ya mencionada naturaleza procedural de la máquina, hay un argumento crítico a favor de la representación procedural del conocimiento: la representación gráfica de un modelo procedural es unívoca, mientras que la representación de un modelo declarativo entraña problemas de significado, precisamente por su cercanía al lenguaje natural. Una prueba de esto es la gran variedad de nomenclaturas que aparece en los trabajos que usan modelos declarativos.
En cuanto a la flexibilidad y la facilidad de acceso al conocimiento que ofrece la representación declarativa, se puede objetar que el conocimiento se maneja de forma local, por lo que no es fácil implantar modelos generales estructurados. En esta tesis se propone el desarrollo de modelos estructurados, resolviendo el problema de la

complejidad con ayuda de lenguajes gráficos. La estructura en red de árboles, por ejemplo, permite representar cualquier modelo con el grado de flexibilidad deseado sin perder el control.
En [Rehak et al, 85] se presenta un sistema basado en una estructura de "macrofunciones"; en [Struss, 87] se define un "structure to function approach" en el que se explota la modularidad de una estructura compleja de funciones.
En cuanto a la relación entre reglas y funciones, en [Kramer, 87] se transforma un modelo de reglas lógicas en un modelo integrado por funciones matemáticas como forma de abordar problemas. En [Kamal et al, 87] se usan "special rules called procedures".
En [Grossman, 90] se indica que los sistemas de reglas pueden siempre modelarse como problemas de variables discretas (en particular variables 0-1) con restricciones lineales:

> "...it has been recently shown by a number of authors...that virtually any propositional statement can be systematically translated into a set of linear inequalities involving 0-1 variables."

2.3 - Desarrollo Sistemático de un Proyecto de Ingeniería del Conocimiento.

En las carreras técnicas en general los estudiantes son evaluados fundamentalmente por su capacidad de aplicar conocimientos teóricos al modelado de situaciones nuevas, es decir, formalizar y resolver numéricamente problemas enunciados textualmente. Un **proyecto de Ingeniería del Conocimiento** no es más que un problema, planteado a profesionales con experiencia, para la resolución del cual se dispone típicamente de varios meses y de toda la documentación que se considere necesaria.

El método que se expone a continuación tiene su origen en la experiencia adquirida al trabajar en proyectos reales de optimización de diseño por ordenador. Estos proyectos han sido lo suficientemente complejos como para requerir la aplicación de un método sistemático en su desarrollo, método que se ha generalizado para ser aplicado a un problema cualquiera.

Una de las dificultades clave en estos proyectos es **obtener el conocimiento** de aquellas personas expertas en el problema que se desea resolver. Dichas personas son capaces de responder a preguntas muy concretas sobre sus conocimientos, pero a veces no es fácil plantear ni tampoco responder preguntas generales que proporcionen una estructura de base al problema.

La segunda gran dificultad consiste en **realizar un modelo** adecuado del problema y de su proceso de resolución. Cuando el problema es suficientemente complejo, el modelado puede dar lugar a un número muy grande de subproblemas y de datos interrelacionados que hay que mantener siempre bajo control.

Por último, la **implantación informática** del sistema destinado a resolver el problema puede ser una tarea muy complicada, tanto en su planificación y desarrollo como en la validación sistemática y búsqueda de errores.

Nota bibliográfica:
En [McFarland et al, 90] se proponen tres tareas muy similares a éstas para el diseño de sistemas digitales: 1) Behaviour (conocimiento sobre el problema), 2) Structure (modelado, algoritmo), 3) Physical Design (implantación). Esto confirma la idea de que el diseño de software presenta grandes similitudes con el diseño de hardware, a pesar de la diferencia evidente de nivel de abstracción.

A lo largo de este apartado se verá cómo estas tres grandes tareas se pueden afrontar con más garantías de éxito siguiendo un método sistemático que se apoya, fundamentalmente, en un buen sistema de representación gráfica del conocimiento.

2.3.1 - Adquisición del Conocimiento y Modelado de un Problema.

El desarrollo sistemático de un proyecto de Ingeniería del Conocimiento comprende tres etapas: adquisición del conocimiento, modelado e implantación. El proceso conjunto está representado en forma de algoritmo en la figura 2.1. Todos los procesos tienen en común el generar una representación formal de sus resultados (llamada "Represent." en la figura 2.1).

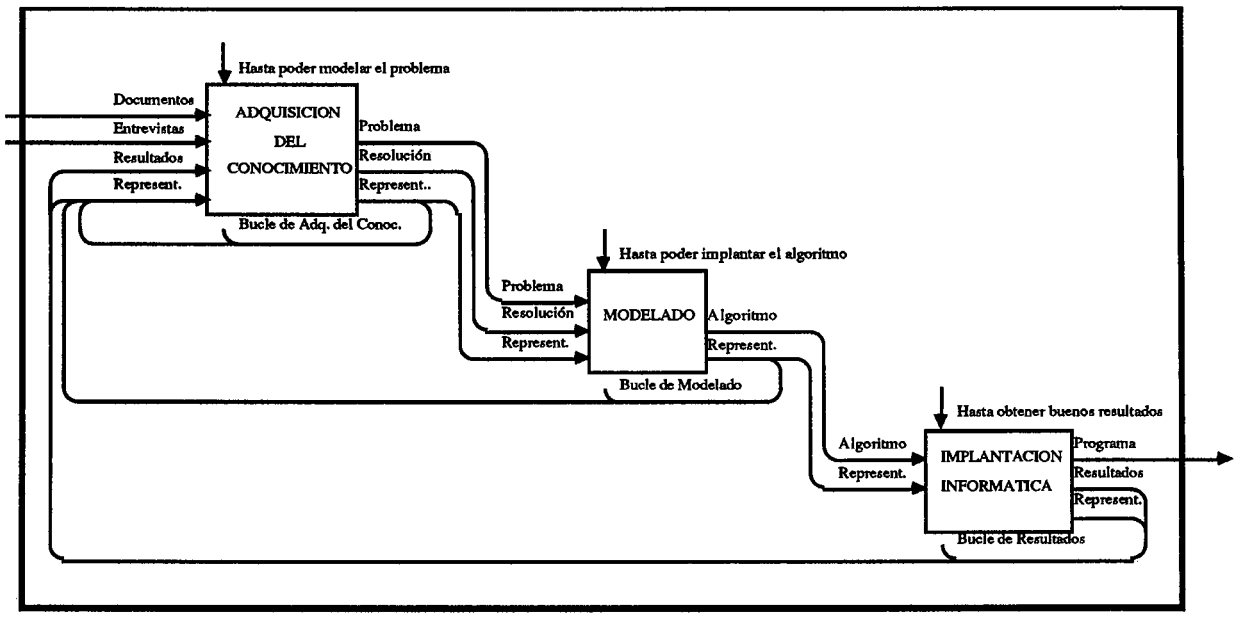


Figura 2.1. Desarrollo Sistemático de un Proyecto de Ingeniería del Conocimiento

Como ya se expuso en el apartado 2.2.2, un sistema automático de resolución de un problema consiste en la implantación informática de un algoritmo. A su vez, el desarrollo de un algoritmo requiere la adquisición del conocimiento necesario y el modelado tanto del problema en sí como de su resolución.

El desarrollo de un sistema automático de resolución de un problema requiere un conocimiento profundo y preciso del problema que se desea resolver. Es muy frecuente que las personas que poseen ese conocimiento carezcan del tiempo y de la objetividad necesarios para formalizar el problema exhaustivamente. Es aquí donde interviene el ingeniero del conocimiento, que deberá contar con una base suficiente en disciplinas generales (Matemáticas, Física, Economía, etc.) y una fuerte base en Lógica aplicada, a ser posible unidas a la experiencia y la creatividad.

La **adquisición del conocimiento** sobre las variables y los procesos que comprende la resolución de un problema es una tarea llevada a cabo por el ingeniero del conocimiento (abreviado IC) en colaboración con el experto en el problema. Esta tarea se puede facilitar enormemente con la ayuda de un método gráfico de representación formal, siguiendo el siguiente método sistemático de aprendizaje y representación formal:

i) El experto proporciona **documentación** general y ejemplos sobre el problema al IC; el IC proporciona al experto documentación general y ejemplos del método gráfico de representación usado.

ii) El IC revisa la documentación y trata de **modelar** el problema, **representándolo** según el método gráfico. El modelado será en general incompleto y revelará lagunas de información y defectos de comprensión en el conocimiento manejado por el IC.

iii) El experto proporcionará **documentación adicional** y aclarará las **dudas** existentes sobre aspectos concretos. El IC explicará el modelado propuesto, discutiéndose posibles mejoras y ampliaciones. La representación gráfica de las distintas versiones del algoritmo queda como testimonio de los avances y mejoras efectuados.

iv) Se repite el punto (ii) hasta encontrar una versión definitiva, pudiéndose entonces comenzar la fase de **implantación**. En muchos casos la fase de implantación impondrá o sugerirá cambios en el algoritmo, que serán discutidos conjuntamente y reflejados en la representación gráfica.

Como puede verse, este proceso sistemático comprende los tres bucles representados en la figura 2.1: el bucle de adquisición, el bucle de modelado y el bucle de implantación. En todos ellos la representación formal sirve de hilo conductor. El sistema de representación formal permitirá también la transmisión del algoritmo y su especificación informática, facilitando la implantación. Jugará el papel que juega el conjunto de planos que es imprescindible seguir en el diseño y construcción de cualquier sistema complicado de Ingeniería o Arquitectura.

En las sucesivas etapas de adquisición del conocimiento y de modelado, la representación formal del problema (y su método de resolución) irá transformándose paulatinamente. Comenzará con descripciones de grandes bloques para las funciones y nombres largos y descriptivos para las variables, y acabará asignando nombres concretos a las variables y entrando en detalles de arquitectura de programación (como puede ser la estructuración de datos y la gestión de memoria).

El **algoritmo** que se implantará en la máquina es la **representación formal** de la función que resuelve el problema, o más exactamente, de la **ley de formación de valores** de dicha función. Recordando que un algoritmo es un conjunto estructurado de procesos y símbolos, se llama aquí **modelado** del problema al proceso de creación del algoritmo encargado de resolverlo.

La definición de **modelo** que aparece en [Larousse, 67] es:

> "Toda estructura lógica o matemática que se utiliza en la Ciencia para dar razón de un conjunto de fenómenos relacionados entre sí."

Modelar un problema es por tanto descomponerlo en un conjunto estructurado de subproblemas; cada subproblema es formalmente otra función vectorial de variable vectorial, y se relaciona con los otros subproblemas por intercambios de datos (variables) y por las secuencias en que deben ser resueltos. Cada subproblema se modelará a su vez, dando lugar a otro conjunto de subproblemas más elementales. El proceso de descomposición terminará cuando todos los subproblemas no modelados sean suficientemente elementales o ya conocidos.

El método sistemático que se sugiere para ello es claramente racionalista: consiste en someter el modelado del problema a una sucesión de procesos de análisis y síntesis. Así describe [Larousse, 67] los procedimientos de **análisis** y **síntesis** intelectuales:

> "Procedimientos generales del pensamiento consistentes en descomponer mentalmente un concepto, un juicio, un raciocinio en sus elementos, y en recomponer seguidamente el todo. Tales son los procedimientos que recomienda Descartes en el segundo y tercer preceptos de su método."

Al comenzar el modelado se cuenta con unas variables de entrada que representan los datos del problema, unas variables de salida que representan las soluciones y una función que relaciona ambos vectores de variables y que representa el proceso de resolución del problema. Al final del modelado se deben obtener dos estructuras estrechamente relacionadas: la estructura de funciones y la estructura de variables. Más adelante se verá que ambas estructuras son análogas y que su forma se puede describir como una "red de árboles".

El método general sugerido para modelar un problema, o lo que es igual, para desarrollar un algoritmo, consiste en el siguiente proceso de **análisis y síntesis**:

i) Se analiza la función global del problema, descomponiéndola en funciones más sencillas; cada una de estas funciones se analiza a su vez, hasta llegar a obtener funciones elementales o ya conocidas.

ii) Se estudia qué variables tiene cada función elemental como argumentos y qué variables son los valores (resultados obtenidos) de cada función.

iii) Se procede a la síntesis (unión, composición) de datos y funciones, buscando la coherencia del conjunto y agrupando los datos en árboles para que cada función, a cualquier nivel del árbol de funciones, tenga un número razonable de argumentos y valores.

iv) Los pasos (ii) y (iii) pueden revelar por un lado la necesidad de funciones auxiliares para transformar o inicializar variables, y por otro la existencia de funciones comunes y vectores de datos comunes en distintas partes del algoritmo. Estas circunstancias dan paso a un segundo proceso de análisis-síntesis, y así sucesivamente hasta conseguir un modelo completo y satisfactorio del algoritmo (ver "bucle de modelado" en la figura 2.1).

La tarea del Ingeniero del Conocimiento es el diseño e implantación de algoritmos capaces de resolver problemas, siguiendo cualquier método sistemático. Aquí se propone utilizar un sistema gráfico de representación del conocimiento para facilitar el proceso descrito, desde la

adquisición del conocimiento hasta la implantación informática. En el apartado 2.4 se verá que el sistema de representación del conocimiento utilizado en todo el proceso puede ser a la vez un sistema de representación de programas (software), con lo que el desarrollo de un algoritmo producirá a la vez la especificación informática correspondiente.

2.3.2 - Algoritmo: Variables, Funciones y Estructura de Red de Arboles.

En la figura 2.1 se aprecia que los productos de las fases de adquisición del conocimiento y de modelado son un algoritmo y su representación gráfica. En este apartado se describe con cierto detalle el concepto de algoritmo y su estructura general.

Se pretende que la máquina sea capaz de resolver un problema. Ahora bien, la resolución de un problema es formalmente una **función vectorial de variable vectorial** definida entre los datos y las soluciones, cuya ley de obtención de imágenes se expresa por medio de un algoritmo. Por ejemplo, en [Veinott, 72] se caracteriza un problema de Ingeniería como sigue:

> "An engineering problem has three basic parts, or elements: (1) given data, (2) answers and (3) a procedure that has to be followed in order to obtain the answers from the given data."

En [Sacristán,64] se describe así el concepto de **función** lógica:

> "...es una operación que consiste en correlacionar dos campos de valores, llamados respectivamente 'de los argumentos' y 'de los valores', de tal modo que a cada individuo del primer campo corresponda por la función un individuo, y sólo uno (correspondencia unívoca), del segundo."

El concepto matemático de función encaja plenamente con la definición anterior. En [Kuratowsky, 66] se define así la función matemática:

> "El concepto de producto cartesiano nos permite definir el concepto de función de una forma completamente general... Sean X, Y dos conjuntos dados. Como función cuyos argumentos recorren el conjunto X (dominio) y cuyos valores pertenecen al conjunto Y (rango) entendemos un subconjunto f del producto cartesiano XxY con la propiedad de que para cada x perteneciente a X existe un elemento y perteneciente a Y, y sólo uno, tal que <x,y> pertenece a f ."

En el proceso de resolución de un problema a cada valor particular del vector formado por los argumentos (datos del problema, variables de entrada, conjunto origen) le corresponde un valor del vector valor de la función (conjunto imagen), formado por una o varias soluciones del problema. El proceso de resolución de un problema es por tanto una función definida entre las variables de entrada y las de salida. La definición de **variable** en Lógica es, según [Sacristán, 64]:

> "Una variable es un símbolo que puede ser sustituido por diversos valores numéricos de una determinada clase."

En cuanto a las variables, la definición de función entre conjuntos es más general que entre variables numéricas, pero en la práctica todo conjunto puede siempre representarse por una variable vectorial tal que a cada elemento del conjunto le corresponda un valor único de la variable y viceversa.

Se observará que el concepto de variable vectorial y el de función vectorial son suficientemente generales como para representar cualquier procedimiento de resolución de problemas. Los vectores tendrán los campos o elementos necesarios para almacenar toda la información que, sobre los entes que intervengan en el problema, sea utilizada en la resolución del mismo. Estos entes podrán ser de carácter abstracto, como por ejemplo un "escenario económico" o unos "factores de peso de restricciones", o bien objetos reales como un motor o una central eléctrica. Según [Wirth, 80]:

> "La información utilizable por el computador consiste en una selección de datos de la realidad, precisamente el conjunto de datos que se considera relevante para el problema de estudio."

No hay ninguna limitación teórica al nivel de abstracción en los razonamientos, pues las propias funciones pueden ser variables de entrada y de salida de otras funciones de "orden" superior: no hay que olvidar que en el ordenador las funciones o procesos se identifican por una variable, al igual que los datos. Al final del apartado 2.5 se trata ampliamente la relación entre la abstracción y la posible "creatividad" de una herramienta informática.

Las dos estructuras resultantes, la de variables y la de funciones, adoptan una forma general análoga: la de una **red de árboles**, esto es, la de un conjunto de árboles que

comparten ciertas ramas. La figura 2.2 muestra un caso de tres árboles organizados según esta estructura.

La red de árboles de funciones tiene un sólo nodo raíz, que es la función que resuelve el problema. Esta es la función principal o, una vez implantada, el programa principal. Sin embargo, la red de árboles de variables tiene al menos dos nodos raíz, correspondientes a las variables de entrada y a las variables de salida. En la práctica se suelen considerar más de dos árboles de variables, pues la naturaleza del problema suele sugerir una clasificación inicial razonable de los datos.

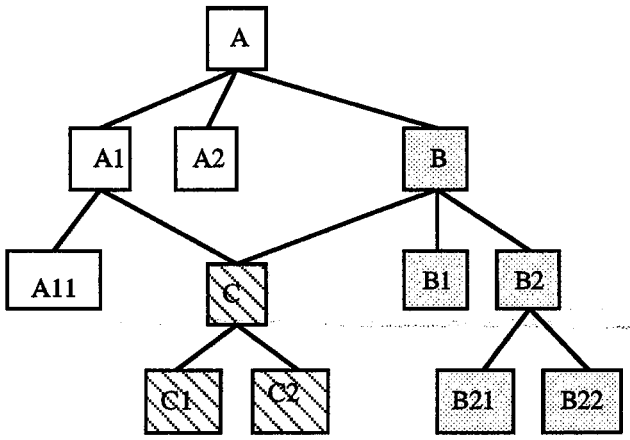


Figura 2.2. Estructura de Red de Arboles

Un árbol de funciones está formado por un conjunto de ellas que sólo son llamadas por una función (son subproblemas de un sólo problema, tienen un sólo nodo padre), a excepción de la función principal (nodo raíz) que es llamada por más de una función, formándose así la red de árboles. En lenguaje informático clásico un árbol de funciones se suele llamar subrutina.

Un árbol de variables está formado por un conjunto de vectores que son componentes de un sólo vector (estructuras de datos que son parte de una sola estructura de datos de nivel superior), a excepción del vector (estructura) raíz del árbol, que aparece como componente de dos o más vectores de nivel superior, formándose así la red de árboles.

Nota bibliográfica:
En [Harrison et al, 90] una red de árboles (no cíclica) se denomina simplemente hierarchical description. En [Gupta, 88] aparece una estructura compuesta de "main trees" y "subsidiary trees".
La estructura de red de árboles de funciones aparece en [Brayton et al, 90] como forma de representación de los circuitos integrados VLSI; según el nivel de detalle en el modelado de los circuitos, la red de árboles de funciones recibe en [Brayton et al, 90] los nombres de Logic Blocks, Network o Boolean Network. La diferencia

fundamental consiste en que en hardware no es posible la recursión (un circuito no puede ser parte de sí mismo) por lo que las redes de árboles en estas aplicaciones son acíclicas.

En la red de árboles de funciones, compartir la misma función supone un ahorro de código de lenguaje de programación y una simplificación en la estructura que aumenta la robustez del algoritmo. En el árbol de vectores de datos la ventaja de compartir ramas es evidente, dado el ahorro de memoria que representa el no duplicar la información sobre un mismo ente. Representa además explícitamente el hecho de que la información manejada al resolver un problema complejo no se puede clasificar o dividir de una forma excluyente, porque los conjuntos de datos suelen presentar intersecciones importantes.

El modelado de un problema puede dar lugar a estructuras muy complejas de funciones y de variables. La utilidad de un sistema gráfico de representación formal salta a la vista, especialmente si se tiene en cuenta que el modelado es una producción sucesiva de versiones cada vez más refinadas de un algoritmo, lo que exige unos medios ágiles y fiables para mantener la coherencia de cada nueva versión.

2.3.3 - Implantación Informática.

Una vez creado un algoritmo capaz de resolver un determinado problema, se podrá proceder a su **implantación informática**. La implantación informática de un algoritmo dado depende naturalmente de las características de dicho algoritmo; a su vez, la implantación informática debe tenerse en cuenta a la hora de concebir el algoritmo si se desea obtener buenos resultados. En cualquier caso, la implantación se enfrenta a tres problemas fundamentales: las limitaciones de tiempo, las limitaciones de memoria y la aparición difícilmente evitable de errores de programación.

La aparición de errores de programación se puede reducir con ayuda de una buena representación formal del algoritmo, a la vez que se facilita su posterior depurado. Los problemas de tiempo y memoria surgen de las limitaciones de las máquinas, y normalmente pueden corregirse introduciendo mejoras en el algoritmo: éste es el significado del "bucle de resultados" que aparece en la figura 2.1.

Nota bibliográfica:
Este control de la máquina es difícilmente alcanzable utilizando modelos declarativos de representación del conocimiento. Precisamente una de las "ventajas" de los modelos declarativos es que el programador no se debe preocupar por el "cómo" ni el "cuándo", sino sólo del "qué". La experiencia indica, sin embargo, que la viabilidad de un proyecto de ingeniería del conocimiento depende muchas veces de cuestiones que sólo pueden resolverse controlando el "cuándo" y el "cómo" de una forma adecuada a cada problema específico.

La implantación se podrá llevar a cabo sistemáticamente comenzando por las funciones elementales; una vez que todas y cada una de las funciones que forman una función más compleja se hayan implantado y comprobado, se integrarán para formar dicha función compleja, generando así progresivamente toda la estructura de funciones.

Cada vez que se integre un grupo de funciones habrá que comprobar los resultados de la función conjunta, si es necesario con herramientas que simulen el comportamiento del exterior frente a dicha función; el tiempo invertido en la creación de estos "bancos de prueba" suele ahorrar largas sesiones de depurado al final del proyecto.

Para que la implantación sea eficaz es preferible que el modelado del problema, y por tanto el algoritmo, se haya orientado hacia una arquitectura informática y un lenguaje determinados. En el campo de la Ingeniería del Conocimiento se encuentran aplicaciones informáticas implantadas en una gran variedad de lenguajes de programación. Es frecuente también que se desarrollen lenguajes a medida para aplicaciones especiales, aunque habitualmente sean versiones especializadas de lenguajes ya existentes.

Nota bibliográfica:
En algunos trabajos se emplea más de un lenguaje de programación, debido a la especialización de las distintas tareas. Entre los lenguajes empleados en aplicaciones al diseño se pueden citar:
- ADA, en [Mac an Airchinnig, 85].
- C, en [Rehak et al, 85], [IIT, 87], [Cuadra & León, 90].
- DSPL, en [Green & Brown, 87].
- FORTRAN, en [Pérez Arriaga, 78], [Cuadra et al, 85], [García et al, 87], [Rehak et al, 85], [Kamal et al, 87].
- IDDL, en [Tomiyama & Ten Hagen, 87].
- LISP, en [Cloarec et al, 85], [Rehak et al, 85], [IIT, 87], [Struss, 87], [Tommelein et al, 87], [Zhang, 89].
- FranzLISP, en [Agogino & Almgren, 87].
- Interlisp-D, en [Dietterich & Ullman, 87].
- ZetaLISP en [Kamal et al, 87].
- MLL, en [Han et al, 87].
- OPS83, en [Maher & Zhao, 87], [Gupta, 88].
- PASCAL, en [Darbyshire & Davies, 85], [Rehak et al, 85], [Kalay et al, 87].
- PROLOG, en [Gero et al, 85], [Szalapaj & Bijl, 85], [Adiba & Nguyen, 85], [Kalay et al, 87].
- PSRL, en [Maher & Fenves, 85].
- SILAGE, en [De Man et al, 90].
- VAXIMA, en [Agogino & Almgren, 87].
- XRL, en [Barbuceanu, 85].

La existencia de tal variedad de lenguajes aplicados a un mismo campo, el de la Ingeniería del Conocimiento aplicada al diseño, muestra la dificultad que entraña pronunciarse con objetividad a favor de un lenguaje de alto nivel. La selección de un "sistema concha" determinado es aún más difícil, puesto que la formalización de cada problema y su resolución son muy subjetivas.

El ser humano utiliza para comunicarse lenguajes textuales (naturales) y lenguajes gráficos. Estos últimos son especialmente importantes en actividades como la Ingeniería, y por tanto también debieran serlo en la Ingeniería del Conocimiento. Por otro lado la máquina opera sólo con variables y funciones (algoritmos) y requiere una gestión eficaz de los recursos de tiempo y memoria, lo que sugiere usar lenguajes de bajo nivel para diseñar el algoritmo a medida de la máquina.

Lo que aquí se propone es una combinación de ambos extremos: diseñar y especificar los algoritmos en un lenguaje gráfico-textual e implantar luego el resultado en un lenguaje de bajo nivel. Dada una especificación completa y consistente, la fase de implantación puede ser una tarea rutinaria, limitándose la creatividad al aprovechamiento óptimo de los recursos de la máquina.

Nota bibliográfica:
Algunos trabajos en los que se utilizan o proponen lenguajes gráficos de programación son [Mac an Airchinnig, 85], [MCC, 89], [Eberling & Wu, 89].

Este combinación de lenguajes gráficos y lenguajes de bajo nivel es especialmente atractiva de cara al futuro, si se piensa que actualmente ya se desarrollan herramientas informáticas que se comunican con el usuario por lenguajes gráficos y son luego capaces de generar código automáticamente en distintos lenguajes de bajo nivel, optimizando incluso los recursos de la máquina automáticamente. La combinación propuesta en esta tesis es la compuesta por la metodología ANA (apartado 2.4) como lenguaje gráfico de alto nivel y el C como lenguaje de bajo nivel (apartado 2.5).

2.4 - Representación del Conocimiento: Metodología ANA.

Ya se ha establecido que es necesario un sistema gráfico de representación formal para la adquisición del conocimiento, el modelado y la implantación de un sistema automático de resolución de problemas. También se ha establecido que el resultado del modelado de un problema es la creación de dos estructuras en forma de red de árboles, la de funciones y la de variables.

El **sistema de representación** elegido debe ser sencillo y completo, capaz de representar fielmente y sin ambigüedad un algoritmo a cualquier nivel de detalle que se desee y

en cualquier nivel de modelado o descomposición en que se encuentre (recuérdese el proceso de análisis-síntesis ya descrito). En otras palabras, y siguiendo la línea de ideas de este trabajo, debe ser capaz de representar cualquier versión tanto de la **red de árboles de funciones** como de la **red de árboles de variables** de un algoritmo. El sistema de representación elegido aquí es el método ANA.

Nota bibliográfica:
El método ANA de representación, surgió como una adaptación del método CORE (Controlled Requirements Expression, [Systems Designers, 85]). Las diferencias entre ambos métodos se deben principalmente a que ANA está especialmente orientado a la representación de software (programas y algoritmos) mientras que el método CORE se desarrolló con una mentalidad de hardware.
En esta tesis se considera que el software (que es el producto final de la Ingeniería del Conocimiento) y el hardware responden a una estructura muy parecida compuesta de variables y funciones, diferenciándose fundamentalmente en el nivel de abstracción de los conceptos que manejan. Hay no obstante unas diferencias prácticas entre software y hardware, que hacen al primero mucho más potente y facilitan su representación gráfica:

i) La recursión, que en hardware tiene que ser simulada por bucles.
ii) El flujo de información, que en hardware tiene que existir físicamente (circuitos que implican consumo, espacio y retrasos), mientras que en software se puede intercambiar una cantidad inmensa de información mediante un único dato (punteros a estructuras y vectores).
iii) La abstracción: en software una función puede ser una variable de entrada o de salida de una función de nivel superior. En hardware es complicado llegar a simular un comportamiento así.

ANA ha sido utilizado ya en proyectos reales y actualmente se cuenta para su aplicación con una herramienta informática gráfica que se ha desarrollado en el transcurso de esta tesis. El método está descrito con detalle en el Apéndice A de la tesis, y será utilizado para describir los algoritmos y ejemplos reales de aplicación.

Para comprobar la capacidad de representación de ANA habrá primero que estudiar con detalle qué tipos de entes y procedimientos pueden aparecer en un problema y por tanto en un algoritmo general. En Lógica se distinguen tres categorías de entes: los individuos, las clases y las relaciones (en [Harrison et al, 90], por ejemplo, se reconocen "entities" o "instances", "entity sets" o "classes", y "relationships"). Estos conceptos son muy intuitivos y, por tanto, tan difíciles de definir como sencillos de utilizar.

Un **individuo** es algo que tiene entidad propia, que puede ser distinguido de otros individuos por llevar ligadas a él una cierta cantidad de propiedades o características. Un individuo podrá ser real o ser abstracto, y en general estará ligado a una cantidad infinita de características (o información) porque puede ser estudiado o considerado desde un número ilimitado de puntos de vista.

Sin embargo, en cuanto a su participación en la resolución de un problema concreto, la cantidad de información que lo caracteriza será finita, es decir, representable por un vector de

variables finito. Dicho vector se puede obtener por síntesis de los vectores de variables de las funciones elementales en las que el individuo participa como argumento o valor.

Todas las posibles ejemplificaciones de un conjunto de propiedades o características forman una **clase** [Stebbing, 65]; un individuo es una ejemplificación particular de la clase a la que pertenece. Los componentes del vector que caracteriza a un individuo serán variables capaces de representar sus propiedades; al tomar dichas variables un valor distinto se obtiene otro individuo de la misma clase.

Se puede representar una clase como un **espacio vectorial**, y los individuos de esta clase como **puntos** concretos de dicho espacio; las características individuales serían las **coordenadas** del vector. A cada individuo le corresponderá un punto del espacio vectorial asociado a su clase, aunque no todo punto del espacio vectorial representará un individuo real.

El concepto de **relación** es aún más difícil de definir. Según [Stebbing, 65]:

> " Toda deducción depende de las propiedades lógicas de las relaciones. No es posible definir la relación sin usar palabras más o menos sinónimas. Todos reconocemos que los individuos en el Universo no están aislados, sino que guardan diversas relaciones entre sí."

Lo más práctico es reducir el estudio de las relaciones al caso que aquí interesa, esto es, a las relaciones que pueden aparecer en la estructura de funciones y la estructura de variables de un algoritmo. Se distinguen primero dos grupos de relaciones: las estáticas y las dinámicas, que se corresponden con lo que en la literatura especializada se suele denominar conocimiento estático y conocimiento dinámico de un problema.

Las **relaciones estáticas** en un algoritmo genérico son las mismas tanto para la estructura de variables como para la estructura de funciones, ya que son simplemente las propias de la teoría de conjuntos (cada conjunto es un nodo de la red de árboles) o del álgebra de clases: Identidad, Pertenencia, Complementariedad, Unión e Intersección. El método ANA es capaz de representar sin dificultad todas ellas para los dos tipos de estructuras.

Las **relaciones dinámicas** son más complejas de enumerar y definir. Son distintas para la estructura de variables y la estructura de funciones, y tal vez las listas de relaciones dinámicas que se presentan a continuación no sean exhaustivas.

En lo referente a la **estructura de funciones** se pueden enumerar las siguientes relaciones dinámicas, que definen los mecanismos de control según los cuales se aplica en cada momento una de las diversas subfunciones que forman una función:

- **Secuencia**, u orden relativo en que se aplican las diversas funciones.

- **Iteración**, o repetición de una función o grupo de funciones hasta cumplir un cierto objetivo.

- **Recursión**, cuando al modelar un problema determinado éste vuelve a aparecer en su misma forma, y así sucesivamente hasta cumplir un cierto objetivo. En la mayoría de los casos la recursión puede sustituirse por una iteración, pero conceptualmente son procesos distintos que a veces no resultan intercambiables.

- **Decisión** de "cuál función se aplica", **condicionada** a la certeza o falsedad de un grupo de asertos. Cuando al modelar un problema aparece este tipo de relación, la función que resuelve el problema se dirá que es una **función redefinida** (término tomado de [Rodríguez-Piñero, 86]).

- **Inicialización**, cuando una función establece las condiciones de partida para que otra función o grupo de funciones pueda aplicarse. Usualmente esto es necesario con procesos iterativos y recursivos, que requieren tanto unas condiciones iniciales como otras condiciones para reconocer el final del proceso.

Con respecto a la **estructura de variables** se pueden distinguir las siguientes relaciones dinámicas, dependientes del flujo de información de un algoritmo y por tanto de la estructura de funciones (por lo que la estructura de variables debe definirse a partir de la de funciones, y no al revés):

- **Causa-efecto** (implicación, argumento-valor, origen-imagen, entrada-salida) entre individuos de la misma o distintas clases, o sea una relación del tipo "ser función de".

- **Instanciación o ejemplificación** de una clase, esto es, tomar un punto concreto de entre todos los puntos posibles de un espacio vectorial. Como casos particulares de ejemplificación pueden citarse la **inicialización** o selección inicial, y la **sustitución** de un individuo por otro de la misma clase, es decir, de un punto de un espacio vectorial por otro punto del mismo espacio.

- **Descomposición** de una clase en subclases, es decir, descomposición de un espacio vectorial en subespacios. La relación inversa será de **composición** o suma.

La metodología ANA es capaz de representar **sin ambigüedades** todas estas relaciones: lo que se puede representar, se puede programar, y viceversa. Utilizando un lenguaje propio de la programación, puede representar el desglose de una estructura (o vector) de variables en sus componentes o elementos, al igual que la operación inversa; la concesión de valores concretos a los argumentos de las funciones o subrutinas; el hecho de igualar un vector o estructura de variables a otro del mismo tipo; y, por supuesto, la relación funcional clásica de "argumentos de entrada - valores de salida" de las funciones.

En cuanto a las secuencias de actuación de las funciones que forman el algoritmo, este método es capaz de representar conjuntos complejos de bucles, bloques "if", recursiones e inicializaciones, mezclados naturalmente con conjuntos de acciones consecutivas. La gran diferencia con los diagramas de flujo tradicionales consiste en que todas las relaciones dinámicas, tanto las de **variables** como las de **funciones**, aparecen representadas en el mismo diagrama. Ya se verá que esto, junto a la absoluta modularidad de la representación ANA, representa un salto cualitativo importante para la comprensión, depurado y desarrollo sistemático de los algoritmos.

La representación ANA de un algoritmo es en principio independiente de su implantación, pues refleja sólo la estructura de subproblemas y la estructura de variables que aparece al analizar sistemáticamente dicha estructura de subproblemas, es decir, representa el modelado de la resolución de un problema en forma de algoritmo. Sin embargo, un modelado que pretenda producir un algoritmo eficaz deberá desarrollarse a medida del estilo y del lenguaje de programación que se vayan a usar en la implantación.

Aunque no se menciona la posibilidad de que varias funciones se apliquen simultáneamente, es decir, no se describe el procesamiento en paralelo, ANA es capaz de representar esto perfectamente. Sin embargo, desde un punto de vista teórico no hay diferencia entre realizar varias acciones en paralelo o secuencialmente, aunque en la práctica signifique un ahorro de tiempo. El procesamiento en paralelo se considera aquí un detalle arquitectural y no determinante de un algoritmo, aunque la posibilidad o imposibilidad de realizarlo puede sugerir a veces diseños radicalmente distintos de un mismo algoritmo.

Entre los dos grandes estilos de programación actuales, la **Programación Estructurada** [Dahl et al, 72] y la **Programación Orientada al Objeto** [Cook, 86] el primero parece encajar más fácilmente con el método de desarrollo y el sistema de representación que aquí se propone. En efecto, la Programación Estructurada está basada en procedimientos modulares organizados según una estructura de red de árboles como la que aquí se describe, por lo que la representación ANA puede usarse directamente como especificación informática de un algoritmo.

La Programación Orientada al Objeto se basa en el concepto abstracto de "**objeto**", siendo cada "**tipo de objetos**" lo que en Lógica se llama **clase**. En el código de los programas escritos en lenguajes de este estilo todas las variables y procedimientos relacionados con un determinado tipo de objetos aparecen agrupados en la declaración de dicho tipo. Esto es bueno en cuanto a mantener la coherencia de cada objeto en su propio entorno, pero en cambio no es fácil reconocer la secuencia de acciones de un programa ni prever las consecuencias de un cambio en esa secuencia.

En Programación Orientada al Objeto, el método ANA sería de gran ayuda para diseñar los algoritmos y ayudar al desarrollo del código, pero en su versión actual no serviría directamente como especificación informática.

2.5 - Lenguaje de Programación C.

Aunque la selección del lenguaje es una decisión personal, se van a exponer aquí razones más o menos objetivas para proponer el lenguaje C como el más adecuado para desarrollar proyectos complejos de Ingeniería del Conocimiento. Se excluyen de la discusión los lenguajes construidos a medida de una aplicación concreta, pues se da por supuesto que siempre serán preferibles en su propio contexto a cualquier lenguaje de aplicación general (por ejemplo, el lenguaje SILAGE para diseño de procesadores digitales de señales propuesto en [De Man et al, 90], o el lenguaje HERREN para el tratamiento de normativas propuesto en [San Gil, 90]).

En [Schildt , 86],[Schildt, 87], [Kernighan & Ritchie, 88] y en [Baker, 89] se comentan diversas razones para preferir el C sobre otros lenguajes, tanto para "Inteligencia Artificial" como para programación convencional (aunque por coherencia con la definición de Ingeniería

de Conocimiento aquí propuesta, no se debería hacer esta distinción). Estas **razones** son básicamente las siguientes:

- El C proporciona todos los **recursos** que necesita el programador más exigente para implantar algoritmos complejos: estructuras de datos definibles por el usuario, memoria dinámica, recursión y manejo de direcciones de memoria como otro tipo cualquiera de variables (punteros).

- En los lenguajes más "rígidos", como el FORTRAN, todas las **ligaduras** (bindings) de las variables con sus campos de memoria se realizan en tiempo de preproceso. En los lenguajes más "flexibles", como el LISP, todas las ligaduras se realizan en tiempo de proceso. En C es el propio programador el que elige con absoluta libertad cuándo se producen y cuándo se liberan estas ligaduras, optimizando así el producto en cuanto a tiempo y memoria.

- El C es actualmente el lenguaje más **compatible** y popular, y con mejores perspectivas para el futuro, a juzgar por su capacidad de **difusión**; y son evidentes las ventajas que siempre acompañan al hecho de unificar lenguajes.

- En la práctica, muchos **compiladores** de lenguajes tales como LISP y Prolog están escritos en C, con la ineficacia propia de tener que traducir un estilo declarativo a otro procedural (por ejemplo, de Prolog a C).

- Los paquetes de programas auxiliares, tales como editores, gráficos e interfaces están escritos en C, lo que acarrea frecuentes problemas de **encadenado** si el programa principal no está en C.

- El C proporciona un esqueleto **estructurado** a los programas sin limitar la creatividad, facilitando la consistencia de aplicaciones que además resultan ser de una **rapidez** espectacular.

Las razones anteriores se pueden resumir en la siguiente afirmación: cuando se necesita controlar todos los recursos de la máquina con funciones rápidas, seguras y eficaces se recurre a un lenguaje de bajo nivel como el C. El inconveniente principal del C es precisamente su bajo nivel, es decir, su proximidad al lenguaje máquina y consiguiente falta de parecido con el lenguaje humano, que hacen al código escrito en C difícilmente comprensible.

En defensa de los lenguajes de bajo nivel se puede argumentar que los de alto nivel sólo facilitan una comprensión muy limitada de los programas, pues sigue siendo muy difícil obtener una visión de conjunto contando sólo con el código fuente, por muy de alto nivel que sea el lenguaje. Sin embargo, con un sistema potente de representación como el ANA, se puede facilitar significativamente la comprensión y especificación informática de un programa.

Hay además un argumento realista y más concluyente de cara al futuro por el cual un lenguaje de bajo nivel es preferible: es evidentemente más fácil y eficaz que un ser humano descienda al nivel de lenguaje de la máquina en vez de que ésta ascienda al nivel de lenguaje de un ser humano. La existencia de **bibliotecas** ("libraries") de programas cada vez más sofisticados y agrupados por especialidades de aplicación es la que debe facilitar el trabajo de los programadores, sin limitar en absoluto sus recursos y creatividad personal.

En el campo de la Ingeniería del Conocimiento hay razones adicionales para preferir utilizar lenguajes de bajo nivel. Una de ellas es la **velocidad** de ejecución, que en la resolución de problemas complejos puede significar la rentabilidad o no de la inversión en un proyecto. Otra es la ya mencionada **integración** con herramientas sofisticadas de gráficos y demás interfaces "amigables". Pero la más importante de cara a grandes proyectos es que el grado de **flexibilidad** estructural lo puede escoger el usuario.

Es siempre deseable que el programador intente crear una estructura general rígida y aislar los módulos que realmente necesitan ser flexibles; esto es importante porque la flexibilidad aumenta la dificultad para detectar y corregir errores, y a la vez hace más lenta la ejecución. No debe olvidarse el papel tan destacado que juega la fase de depurado en un proyecto informático.

Nota bibliográfica:
En [Coyne et al, 87] se discute la capacidad de creatividad e innovación de las herramientas informáticas, y en particular las de diseño. La innovación consistiría en hallar nuevos puntos de un espacio ya definido, mientras que la creatividad permitiría definir nuevos espacios y nuevas estrategias de búsqueda. Siempre según [Coyne et al, 87], la creatividad de una herramienta aumenta con su grado de desorden interno (o entropía), aunque otro camino para aumentar la creatividad sea también aumentar el grado de abstracción.
En [Zhang, 89] se propone lograr la creatividad en el diseño automático por medio de analogías en niveles altos de abstracción.
En esta tesis se considera que aumentar el nivel de abstracción de los modelos empleados es el único camino útil para simular la creatividad en una herramienta informática. La idea se basa en que las innovaciones (nuevos puntos) en un espacio de variables correspondiente a un cierto nivel de abstracción pueden representar cambios "creativos" (nuevos espacios, nuevas estrategias) en espacios correspondientes a niveles más bajos de abstracción.
En [Arbab, 89] se distingue entre diseño creativo y diseño paramétrico, según la misma línea de [Coyne et al, 87]. De nuevo se puede señalar que un parámetro puede ser una variable perteneciente a un nivel de abstracción superior, por lo que un cambio en su valor (innovación) puede significar cambios de espacio o de estrategia (diseño creativo) en niveles inferiores de abstracción.
En cuanto a la creatividad y la entropía existe una idea muy extendida acerca de los sistemas de reglas, y en general acerca de los sistemas en los que el conocimiento se añade "pedazo a pedazo": que basta mezclar todos estos pedazos y "agitar" para que se produzcan razonamientos bien encadenados. La realidad es muy distinta: excepto en problemas muy especiales (o muy sencillos) es necesario contar con una estructura de control para

obtener respuestas en un tiempo razonable. Esta estructura de control se puede simular jerarquizando las reglas según niveles de abstracción (meta-reglas), pero debe tenerse en cuenta que es difícil garantizar el control global de un sistema utilizando sólo operadores locales dispersos. Acerca de la flexibilidad y el control de los sistemas de reglas de producción, en [Brayton et al, 90] se dice:

> "Adding a new rule is not simply a matter of adding the rule to the set of transformations; it is also necessary to consider how the new rule will interact with the other rules in the system. In the limit, it may be necessary to rewrite the heuristics which control the order in which the rules are applied."

Como resumen de todas estas consideraciones, y como una de las ideas de fondo de esta tesis, se puede enunciar lo siguiente: lo que se necesita en la Ingeniería del Conocimiento no es un lenguaje sofisticado, sino unas **técnicas de modelado** sofisticadas que puedan generar representaciones suficientemente abstractas de los problemas, auxiliadas por un **sistema gráfico de representación** claro y potente, y a ser posible común para algoritmos abstractos e implantaciones informáticas. El contar con un sistema gráfico de representación formal ha permitido una tarea formidable de creación y desarrollo en disciplinas tan complejas como la Ingeniería y la Arquitectura.

2.6.- Notas sobre Aplicaciones a Proyectos.

Las ideas y metodologías expuestas en este capítulo están inspiradas en el trabajo llevado a cabo en proyectos reales de investigación y desarrollo. En este apartado se comenta el trabajo realizado en dichos proyectos a la luz de las ideas y la nomenclatura utilizadas aquí, como si éstos fueran aplicaciones de la tesis y no al contrario. De esta forma las aplicaciones reales proporcionan un respaldo experimental a las ideas y metodologías expuestas en la tesis. Con este objeto el orden de las notas no es cronológico, sino que se adapta al orden de exposición del capítulo.

En [Cuadra & León, 90] se aplicó una estrategia de solución que combina con éxito la planificación con la búsqueda heurística. En [IIT, 87] se propuso una estructura de control de pizarra para un sistema que combinaba reglas de producción (expresando razonamientos cualitativos) y optimización matemática (según un modelo de razonamiento funcional). Se implantó solamente un prototipo del modelo propuesto, por tratarse de un proyecto de estudio de viabilidad. El modelo de razonamiento funcional se ha utilizado en todas las aplicaciones reales: [Cuadra et al, 85], [García et al, 87], [IIT, 87], [Cuadra et al, 90], [Cuadra & León, 90].

Los pasos propuestos para el desarrollo de un proyecto de Ingeniería del Conocimiento (adquisición del conocimiento, modelado e implantación) se aplicaron sistemáticamente en [IIT, 87] y [Cuadra & León, 90], resultando unas aplicaciones informáticas de gran calidad en campos de aplicación que inicialmente eran totalmente desconocidos para los equipos que los desarrollaron.

En [Cuadra & León, 90] y [Cuadra et al, 90] se crearon sistemáticamente las estructuras de variables y funciones en forma de red de árboles. Esto se llevó al extremo en el desarrollo de las dos herramientas informáticas que acompañan la tesis: la herramienta ANA de representación gráfica de algoritmos y el creador automático de ábacos lineales (a partir de ahora y por motivos prácticos será llamado CAL, Creador de Abacos Lineales, aunque no sea su nombre "comercial").

La conveniencia de desarrollar ambas herramientas informáticas se hizo patente tras el proyecto [IIT, 87]; la metodología ANA se adoptó a partir del estudio de distintos métodos gráficos de representación del conocimiento (en particular el método CORE), y posteriormente la herramienta ANA se desarrolló para hacer posible la utilización sistemática y eficaz de dicha metodología (1989-90). El método de interpolación y la herramienta CAL se desarrollaron en 1988 para resolver un problema de lentitud de cálculo surgido en [IIT, 87] (ver apartado 7.5).

En [IIT, 87], [Cuadra & León, 90] y en el desarrollo de ANA y CAL se siguió un proceso de implantación sistemática y modular según los planos expresados en el sistema ANA de representación. En [IIT, 87], CAL y el propio desarrollo de la herramienta ANA los diagramas ANA se realizaron manualmente, mientras que en [Cuadra & León, 90] y [Cuadra et al, 90] se utilizó ya una versión provisional de la herramienta ANA.

Las listas de relaciones estáticas y dinámicas entre variables y funciones que aparecen en el apartado 2.4 son resultados del proceso de desarrollo de la herramienta ANA.

En cuanto a los lenguajes de programación, se ha empleado el FORTRAN en [Pérez Arriaga, 78], [Cuadra et al, 85] y [García et al, 87]; se ha utilizado el lenguaje C en [IIT, 87], [Cuadra & León, 90] y en las herramientas CAL y ANA; en [IIT, 87] se empleó también el lenguaje LISP (entorno de programación ART, Automatic Reasoning Tool), utilizándose una máquina con dos procesadores, uno de ellos LISP (para ART) y otro UNIX para el lenguaje C.

Capítulo 3

El Problema General de la Optimización de Diseño

En el Capítulo 2 se ha tratado la Ingeniería del Conocimiento en general, como aplicación de la Lógica a la resolución automática de problemas. En este capítulo se explica cómo se puede aplicar lo expuesto al problema central de esta Tesis: la optimización de diseño por ordenador.

En el apartado 3.1 se identifican los elementos básicos del problema revisando la literatura técnica; también se selecciona una nomenclatura única de entre la gran variedad de términos y conceptos que aparecen en las publicaciones estudiadas.

En el apartado 3.2 se presenta un ejemplo de aplicación extraído de proyectos reales de optimización de diseño. El ejemplo se ha simplificado para mostrar las características esenciales del problema de diseño sin entrar en detalles innecesarios.

En el apartado 3.3 se propone una formalización matemática completa del problema de optimización de diseño, ilustrada con su aplicación al ejemplo del apartado 3.2.

El apartado 3.4 describe el enfoque propuesto en esta tesis para la optimización de diseño por ordenador. Este enfoque se denomina **Optimización Estructurada Multiatributo**, y sus distintos aspectos se describen con más detalle en capítulos sucesivos.

Finalmente, el apartado 3.5 está dedicado a comentar aspectos relacionados con este capítulo de los proyectos reales de diseño por ordenador que han dado lugar a esta tesis.

3.1 - Elementos del Problema.

Identificar los elementos de la optimización de diseño por ordenador en la literatura técnica es enfrentarse a una gran confusión de términos y conceptos. Esto es debido en parte a la relativa novedad del tema de estudio y en parte a la mezcla de tres tipos muy distintos de lenguaje:

i) El lenguaje de la industria y de los ingenieros de diseño.
ii) El lenguaje de la Ingeniería del Conocimiento.
iii) El lenguaje matemático relativo a la optimización.

En este apartado se repasarán los conceptos y procesos fundamentales de la optimización de diseño y los distintos términos con los que se identifican dichos conceptos; la nomenclatura adoptada en esta tesis es básicamente la del lenguaje propio de la optimización matemática, con algunos términos adicionales tomados del entorno del diseño.

El diseño de un aparato o sistema consiste en determinar un conjunto de características del mismo, que lo definen por completo para los objetivos que se hayan propuesto. Cada una de estas características se representa por medio de una **variable de diseño**. Las variables de diseño pueden representar magnitudes físicas (dimensiones, pesos, temperaturas) tanto como magnitudes "artificiales" (costes, fiabilidad).

Nota bibliográfica:
En las publicaciones de optimización de diseño por ordenador las variables de diseño se conocen por varios nombres: variables en [Ishii & Barkan, 87] y [Kamal et al, 87]; attributes en [Tomiyama & Yoshikawa, 85], [Tomiyama & Ten Hagen, 87]; parameters en [Kamal et al, 87]; description of an artifact en [Arbab, 89].

Las variables de diseño están relacionadas unas con otras por las ecuaciones (igualdades) propias del modelo del sistema que se haya seleccionado. Estas ecuaciones se denominan aquí **ligaduras**, y típicamente expresan relaciones físicas fundamentales entre las variables de diseño que son impuestas por la naturaleza del aparato o sistema a diseñar. Las ligaduras reducen el número de grados de libertad del diseño, o sea, el número máximo de **variables independientes** que se pueden seleccionar.

Nota bibliográfica:
En los textos de optimización matemática (como [Gill et al, 81] y [Borrel, 87]) no se suele distinguir entre variables independientes y dependientes, quizá porque las restricciones de igualdad se consideran en general no despejables, manejándose como restricciones y no como ligaduras.

En la literatura técnica, las ligaduras se conocen como analysis calculations en [Veinott, 72]; equality constraints (o restricciones de igualdad) en [Gill et al, 81], [Borrel, 87], [Agogino & Almgren, 87], [Ishii & Barkan, 87] y [Kermanshahi et al, 90]; interactions en [Ishii & Barkan, 87]; analysis equations en [Grossman, 90].
A su vez, las variables independientes del diseño son llamadas variables en [Dietterich & Ullman, 87] y [Galiana & McGillis, 88]; independent variables en [Singh et al, 83] y [Kamal et al, 87]; decision variables en [Ishii & Barkan, 87]; alternatives en [Brown & Chandrasekaran, 85] y [Grossman, 90]; decisions en [Chankong & Haimes, 83] y [Struss, 87]; solutions en [Kalay et al, 87]; parameters en [Maly, 90]; degrees of freedom en [Kuh & Ohtsuki, 90].

Los valores que pueden tomar las variables de diseño están limitados también por un conjunto de inecuaciones (desigualdades) llamadas **restricciones** del diseño. Estas restricciones no reducen en teoría el número de grados de libertad del diseño, pero pueden hacer que la solución sea imposible. Una solución se llama **factible** (feasible) cuando cumple todas las restricciones impuestas al diseño. Un tipo particular de restricción es el **rango** o conjunto de valores posibles de cada variable. Otro tipo de restricciones muy importante es el que impone el conjunto de **normas** (standards) aplicables al diseño (véase [San Gil, 90]).

Nota bibliográfica:
Las restricciones del diseño se asocian a distintos términos según los distintos autores: performance requirements en [Veinott, 72]; constraints en [Singh et al, 83], [Maher & Fenves, 85], [Tommelein et al, 87], [Kamal et al, 87], [Navinchandra & Marks, 87], [Fenves & Baker, 87], [Galiana & McGillis, 88], [Grossman, 90], [Brayton et al, 90] y [McFarland et al, 90]; inequality constraints (o restricciones de desigualdad) en [Gill, 81], [Borrel, 87], [Ishii & Barkan, 87], [Agogino & Almgren, 87] y [Kermanshahi et al, 90]; desired behaviours en [Struss, 87], criteria en [Kuh & Ohtsuki, 90]; hard criteria en [Kalay et al, 87]; limitations en [Tommelein et al, 87]; requirements en [Grossman, 90].
La violación de restricciones se formula de distintas formas: failures en [Brown & Chandrasekaran, 85]; contradictions en [Struss, 87]; deviation variables en [Kamal et al, 87].
Las soluciones que no violan las restricciones se llaman en casi todas las publicaciones revisadas feasible solutions (soluciones factibles), aunque por ejemplo en [Kuh & Ohtsuki, 90] se habla de legal solutions.

Hay variables de diseño que se utilizan como medidas de la calidad de cada solución particular, permitiendo comparar dos soluciones entre sí (sean o no factibles). Estas variables se llaman aquí **atributos** del diseño, y son estos atributos las magnitudes con respecto a las cuales se optimiza. Atributos típicos en diseño son el coste y la fiabilidad; hay también problemas en los que es importante minimizar pesos y dimensiones, maximizar rendimientos, etc.

Nota bibliográfica:
Los atributos del diseño reciben distintos nombres en la literatura técnica: attributes en [Schweppe & Merril, 86], [Kamal et al, 87] y [Coyne et al, 87]; features en [Maher & Fenves, 85]; desired behaviours en [Struss, 87]; objectives en [Tommelein et al, 87], [Agogino & Almgren, 87], [Navinchandra & Marks, 87], [Maly, 90], [Kermanshahi et al, 90] y [McFarland et al, 90]; criteria en [Kermanshahi et al, 90], [Kuh & Ohtsuki, 90] y [De Man et al, 90]; evaluation criteria en [Veinott, 72]; performances, performance aspects o performance levels en [Ishii & Barkan, 87], [Allen, 90] y [Coyne et al, 87]; performance criteria en [Galiana & McGillis, 88]; soft criteria en [Kalay et al, 87]; measures of merit en [Kamal et al, 87]; goals en [Grossman, 90] y [Brayton et al, 90]; interpretations en [Coyne et al, 87].
El espacio de atributos se denomina objective space en [Chankong & Haimes, 83] y [Jazdzynski, 89]; design space en [McFarland et al, 90].

Una **función objetivo** (objective function) es una función escalar que asocia cada solución a una cifra cuyo valor se desea optimizar. Esta función permite comparar unas soluciones con otras de forma absoluta. Si hay más de un atributo a considerar en el problema de diseño se dice que se trata de una optimización multiatributo, y para resolverla hay dos opciones:

i) Se define una única función objetivo que incluya todos los atributos (por ejemplo en forma aditiva), concediendo distinta importancia relativa a cada uno, con lo que se convierte el problema en una optimización monoatributo. El óptimo es en teoría un sólo punto, aunque en la práctica pueda haber más de un óptimo local o incluso óptimos "planos" (zonas en las que la función objetivo varía de forma despreciable).

ii) Se trata cada atributo por separado, llevando a cabo una verdadera optimización multiatributo de acuerdo con procedimientos que se explicarán más adelante. El óptimo en una optimización así resulta ser una hipersuperficie en el espacio de atributos, que normalmente se describirá como un conjunto de puntos representativos de la misma.

Nota bibliográfica:
La función objetivo se conoce con diversos nombres en la literatura: objective function en [Gill et al, 81], [Singh et al, 83], [Tommelein et al, 87], [Ishii & Barkan, 87], [Agogino & Almgren, 87], [Borrel, 87], [Kuh & Ohtsuki, 90] y [McFarland et al, 90]; goal en [Tomiyama & Yoshikawa, 85]; optimization criterion en [Tommelein et al, 87]; merit function en [Kamal et al, 87].
Algunos trabajos en los que se aborda la optimización multiatributo por medio de una única función objetivo son [Maher & Fenves, 85], [Tomiyama & Yoshikawa, 85], [Agogino & Almgren, 87], [Maher & Zhao, 87], [Kalay et al, 87], [Tommelein et al, 87], [Ishii & Barkan, 87], [Kamal et al, 87] y [Appelbaum et al, 87].
Los pesos o importancias relativas que se conceden a los distintos atributos en una función objetivo se conocen como weights o weigthing factors en [Maher & Fenves, 85], [Agogino & Almgren, 87] y [Maher & Zhao, 87]; criteria en [Kalay et al, 87].
La segunda opción, es decir, la optimización realmente multiatributo, se denomina Multivariate Decision Problem (MDP) y también Vector Optimization Problem (VOP) en [Chankong & Haimes, 83]; multicriterio en [Borrel, 87], multivariate objective en [Brayton et al, 90].
La optimización multiatributo considerando cada atributo por separado se aborda en [IIT, 87], [Navinchandra & Marks, 87], [Nanda et al, 87], [Jazdzynski, 89], [Cuadra et al, 90] y [Kermanshahi et al, 90].

En un problema de diseño existe normalmente un cliente al que va destinado el producto final. El cliente impone un conjunto de cualidades que como mínimo debe cumplir dicho producto, y tal vez el deseo de optimizar lo más posible ciertas medidas de su funcionamiento (performance). En otras ocasiones no es un cliente concreto sino la competitividad del mercado la que impone la optimización y las cualidades mínimas del producto. Por otro lado, el producto estará destinado a funcionar en un entorno determinado cuyas características deben conocerse (temperaturas, tensiones, etc.).

Todo esto se traduce en que hay un conjunto de variables de diseño cuyos valores deben fijarse para que el problema quede perfectamente especificado. Estas variables se llaman aquí **requisitos** del diseño; el cambio de valor de uno o más requisitos de diseño hace que cambie el problema. Según la nomenclatura del capítulo anterior, los requisitos son las variables de entrada de la función "resolver el problema de optimización de diseño", así como las variables de salida son las soluciones.

Nota bibliográfica:
Los requisitos del diseño se identifican con términos muy dispares en las publicaciones: requirements en [Brown & Chandrasekaran, 85], [Tommelein et al, 87], [Han et al, 87]; specifications en [Tomiyama & Yoshikawa, 85], [Ishii & Barkan, 87], [Krishnan et al, 88]; functions en [Tomiyama & Ten Hagen, 87]; design parameters en [Galiana & McGillis, 88].
Como ejemplo de estructuración del conocimiento, en [Han et al, 87] un requisito se considera una estructura compleja de información compuesta de "attributes, properties, behaviours and functions". En esta tesis se considera que cada problema de diseño particular requiere su propia estructuración de variables; un requisito puede ser tanto una variable escalar (temperatura máxima) como una función definida por un conjunto de puntos (curva de par/velocidad de un motor).
Un caso especial en la formalización de requisitos es el diseño de circuitos VLSI y ASIC. En estos problemas, aparte de los requisitos habituales relacionados con máximo tamaño, máximo tiempo de respuesta, etc., el requisito fundamental es el código de un programa que describe por completo el comportamiento del circuito a diseñar.

Las variables de diseño (independientes o dependientes), las ligaduras, las restricciones, los atributos y los requisitos del diseño son elementos de cada problema de diseño que sólo se pueden identificar por medio de un modelado de dicho problema. Ahora bien, el diseño de un mismo sistema o aparato se puede modelar con distintos niveles de detalle. Esto permite abordar el problema mediante una serie de etapas sucesivas, llamadas aquí **fases de diseño**, mediante las cuales se va definiendo la solución (o soluciones) con sucesivos niveles o grados de detalle.

En la mayoría de las publicaciones sobre optimización de diseño por ordenador se propone un cierto número de fases de diseño y un nombre más o menos afortunado para cada una de ellas. En general se limitan a aplicar términos usados en los procesos industriales de diseño que resultan de poca utilidad en una herramienta informática, pues la separación entre fases responde más a criterios de gestión de proyectos que a diferencias conceptuales.

Nota bibliográfica:
En la mayoría de trabajos se proponen unas fases de diseño muy similares: en [Tomiyama & Yoshikawa, 85], [Tomiyama & Ten Hagen, 87]:
1) conceptual design. 2) basic design. 3) detail design.
En [Brown & Chandrasekaran, 85]:
1) requirements. 2) rough design. 3) design. 4) redesign.
En [Rehak et al, 85]:
1) predesign. 2) conceptual design. 3) detailed design. 4) completed detailed design.
En [Dietterich & Ullman, 87]:
1) specification. 2) conceptual design. 3) qualitative analysis. 4) detailed design. 5) quantitative analysis.
En [Han et al, 87]:

1) requirements. 2) conceptual design. 3) preliminary design. 4) detail design.

En [Maher & Zhao, 87]:

1) preliminary (conceptual) design. 2) analysis. 3) detailed design.

En [Zhang, 89]:

1) determine design specifications. 2) conceptual (preliminary) design. 3) layout design. 4) detail design.

Otras divisiones en fases de diseño están muy impuestas por los algoritmos empleados y la aplicación específica.

En [Bowen, 85]:

1) interviews (specification). 2) design. 3) report.

En [Murthy & Addanki, 87]:

1) label propagation. 2) modification operators. 3) changes from equations.

En [Agogino & Almgren, 87] se plantea un proceso de diseño dividido en tres fases, cada una de las cuales se aborda con un enfoque totalmente distinto (lo que parece una idea original e interesante):

1) qualitative design (resuelta por razonamiento cualitativo).
2) functional design (resuelta por análisis de monotonía).
3) numerical design (resuelta por optimización matemática).

En el contexto del diseño de circuitos lógicos VLSI (llamado en general silicon compilation) se describen unas fases de diseño especiales, ampliamente aceptadas y muy definidas, correspondientes a niveles concretos de detalle en el modelado. La razón de este consenso estriba en lo avanzado (gran experiencia) de dicho campo de aplicación, impulsado indudablemente por los intereses económicos de la industria. Tomando como base [Allen, 90] y [Kuh & Ohtsuki, 90], las fases de diseño de circuitos VLSI y sus niveles de detalle son:

1) Behavioural Synthesis, correspondiente al nivel de detalle "functional & behavioural"
2) Register-Transfer Synthesis, correspondiente al nivel "structural or architectural".
3) Logic Synthesis, correspondiente al nivel de "logical functions" (en [Brayton et al, 90] se distinguen tres subniveles de detalle: a) logic blocks, b) networks, c) boolean networks).
4) Circuit Synthesis, correspondiente al nivel de "mask geometries". A partir de aquí, todo se considera a nivel "mask geometries" pero con distinto nivel de detalle.
5) Layout Synthesis, que se puede dividir en subfases: a) partitioning, b) floorplanning, c) placement, d) global routing, e) detailed routing.

En [McFarland et al, 90] se reconocen más o menos las fases anteriores, aunque no todos los términos propuestos se correspondan uno a uno con los de dichas fases:

1) Processors, Memories and Switches.
2) Algorithm.
3) Register-Transfer.
4) Logic.
5) Circuit.
6) Devices.

Especial atención merece la implantación de una fase inicial de carácter interactivo y tutorial destinada a verificar la coherencia y validez de los datos proporcionados por el usuario, especialmente en problemas de diseño muy complejos. Esta fase se llamará aquí "**análisis de consistencia y prediseño**", aunque en la literatura técnica aparece con nombres diversos (ver apartado 3.4.2).

La separación en fases de diseño es una idea muy útil para la resolución automática del problema. Sin embargo, el número de las fases y su nombre dependerá de cada aplicación particular, pues los distintos niveles de detalle seleccionados en el modelado de un problema no son generalizables a otro; de hecho pueden depender de la velocidad de cálculo propia de cada proceso de evaluación. En el apartado 3.4 se expondrá cómo se propone abordar en esta tesis la descomposición de un problema en fases de diseño.

Cada fase de diseño consiste en hallar un conjunto de soluciones interesantes, que serán empleadas como puntos de partida para la siguiente fase de diseño (mayor nivel de detalle en el modelado), y así sucesivamente hasta alcanzar soluciones con el máximo nivel de detalle deseado. Otra misión de cada fase es limitar la zona de búsqueda de la fase siguiente, pues cada fase consistirá en un conjunto estructurado de procesos de búsqueda o exploraciones llevadas a cabo en el espacio de variables independientes.

Ya se habló en el capítulo anterior de las dos grandes familias de procesos de búsqueda de soluciones en la Ingeniería del Conocimiento: la búsqueda heurística (muchos estados evaluados, poco control en los operadores) y la planificación (menos estados evaluados gracias a un control más sofisticado). En el entorno de la optimización de diseño es aplicable esta clasificación, siendo los estados soluciones provisionales y siendo los operadores las modificaciones realizadas para mejorar las características de las soluciones.

El ingeniero lleva a cabo la búsqueda de soluciones mediante un procedimiento iterativo y más o menos sistemático de prueba y error: se plantea un diseño inicial, se analiza (o evalúa) y se efectúan cambios para generar un nuevo diseño, que será a su vez analizado y retocado. Este tipo de proceso será denominado a lo largo de este trabajo **bucle de diseño**, y consta de las siguientes tareas:

- Conociendo los requisitos impuestos, plantear buenos **diseños iniciales**.

- Conociendo los resultados arrojados por la evaluación de un determinado diseño, proponer **cambios** en las variables independientes que produzcan otro presumiblemente mejor.

- Conociendo la marcha del bucle de diseño, decidir cuándo puede darse por **terminada** la búsqueda en curso, teniendo en cuenta siempre el riesgo de conformarse con óptimos locales por no explorar las zonas que conducen al óptimo global.

Estas tareas se pueden resumir en dos: la **evaluación** de soluciones y la **estrategia de búsqueda**. El ingeniero tratará de ahorrar tiempo evaluando el menor número posible de soluciones, basándose en una buena estrategia de búsqueda inspirada por su experiencia, sus conocimientos y su intuición. Los resultados ofrecidos por el ordenador se apoyarán más en la evaluación sistemática de un gran número de soluciones, aunque será siempre de gran utilidad poder implantar estrategias de búsqueda eficaces.

Nota bibliográfica:
El bucle de diseño aparece en gran número de publicaciones, si bien con distintos nombres: design procedure en [Veinott, 72]; design process en [Tomiyama & Yoshikawa, 85]; trial and error en [Ishii & Barkan, 87]; iterative redesign process en [Brown & Chandrasekaran, 85] y [Ishii & Barkan, 87].
El bucle de diseño se presenta compuesto por distintas tareas: 1)initial design geometry, 2)analysis calculations, 3)evaluation of results, 4)development of next trial design, en [Veinott, 72]; 1)goal ellaboration, 2)design generation, 3)design evaluation, en [Barbuceanu, 85]; 1)analysis, 2)synthesis, 3)evaluation, en [Gero et al, 85]; 1)tentative model, 2)evaluate, 3)modify and back to (2) until feasible design, en [Han et al, 87]; 1)specify the desired behaviours, 2)design decisions, 3)simulation, 4)detect contradictions, 5)determine the underlying assumptions and go back to (2), en [Struss, 87].
La búsqueda recibe distintos nombres según el modelo seguido para el proceso de diseño (obsérvese la notación propia de los modelos lingüísticos): selection en [Maher & Fenves, 85], generation en [Barbuceanu, 85]; analysis and synthesis en [Gero et al, 85]; modifications en [Han et al, 87]; exploration en [Allen, 90].
Lo mismo ocurre con la evaluación: se llama evaluation en [Tomiyama & Yoshikawa, 85], [Barbuceanu, 85], [Gero et al, 85] y [Han et al, 87]; analysis en [Rehak et al, 85] y [Maher & Zhao, 87]; simulation en [Struss, 87] y [Ishii & Barkan, 87]; performance en [Tommelein et al, 87]; meaning en [Coyne et al, 87]; verification en [Allen, 90].

En cuanto a las estrategias de búsqueda, se tratarán con detalle en el Capítulo 5. Sin embargo, se puede anticipar aquí la clasificación de las estrategias en dos grandes familias: la **búsqueda local** y la **búsqueda en árbol**. La primera es más adecuada para espacios de variables continuas y problemas monoatributo, mientras que la segunda es adecuada para problemas con variables discretas y facilita la optimización multiatributo.

Nota bibliográfica:
Algunos trabajos de optimización de diseño en los que se utilizan búsquedas locales son [Pérez Arriaga, 78], [Cuadra et al, 85] y [García et al, 87].
Búsquedas en árbol se pueden encontrar en [Maher & Fenves, 85], [Tommelein et al, 87], [Dietterich & Ullman, 87], [Kalay et al, 87], [Murthy & Addanki, 87], [Cuadra & León, 90] y [Latorre et al, 90].
Se combinan ambos tipos de búsqueda en [Beale, 77] (el ya clásico algoritmo Branch & Bound); en problemas de diseño se pueden citar [IIT, 87] y [Allen, 90]; en [Kuh & Ohtsuki, 90] se revisa un conjunto de estrategias de búsqueda que combinan búsquedas en árbol con búsquedas locales aplicadas al diseño de circuitos VLSI.

Resumiendo este apartado, los elementos de la optimización de diseño por ordenador que se han identificado aquí son los siguientes: variables de diseño, ligaduras, variables independientes, restricciones, atributos, funciones objetivo, requisitos, fases de diseño y bucles de diseño (búsqueda y evaluación). Se verá ahora cómo se puede abordar un problema completo y general de optimización de diseño: en el apartado siguiente se presenta un caso ejemplo real, aunque simplificado, que se utilizará para ilustrar el contenido del resto del capítulo.

3.2 - Caso Ejemplo.

El ejemplo de aplicación que aquí se presenta es una versión simplificada de un problema industrial de diseño estudiado con detalle en [Cuadra et al, 85] y [García et al, 87]. El número de variables independientes y de restricciones se ha reducido aquí lo más posible con objeto de hacer más clara la exposición, manteniendo una muestra de cada aspecto de interés.

Se desea diseñar una **reactancia de núcleo de aire** para ser intercalada en una línea aérea de distribución. La reactancia irá conectada en serie y será atravesada por tanto por la intensidad de la línea. La reactancia está compuesta por varias **bobinas** concéntricas de una sola capa, arrolladas una sobre la otra y conectadas entre sí en paralelo con objeto de repartirse la intensidad de la red. El número de **espiras** de cada bobina deberá ser entero.

El tipo de conductor utilizado en las bobinas es fijo e igual para todas: una pletina de aluminio de sección rectangular y lados **{a,b}**, recubierto de una capa de esmalte aislante. El conductor tiene una elasticidad limitada, por lo que las bobinas tendrán un **diámetro mínimo** admisible. La figura 3.1 muestra un esquema de la reactancia y sus dimensiones principales.

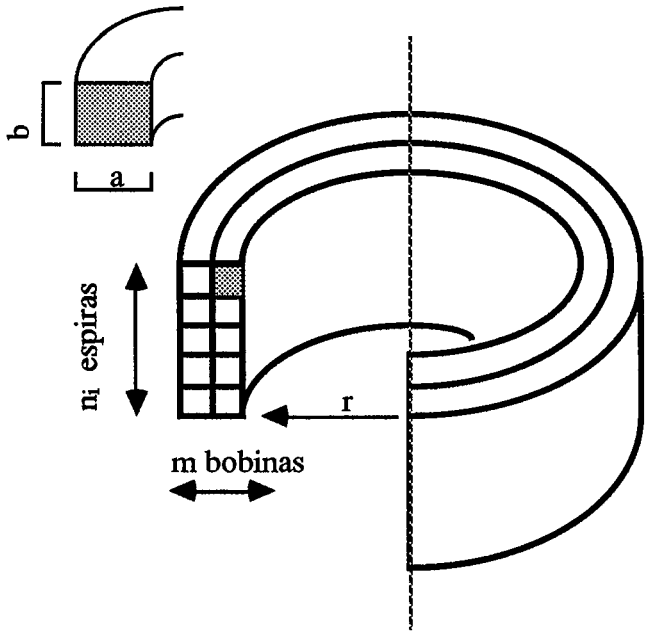


Figura 3.1. Esquema de la Reactancia

Las variables independientes seleccionadas son:

$$X = \{ m, n_1, n_2, ..., n_m, r \} = \{ m, N, r \}$$

donde:

m	número de bobinas en paralelo
$N = \{ n_1, n_2, .., n_m \}$	número de espiras de cada bobina
r	radio interior de la reactancia

Obsérvese que el número de variables independientes es variable, pues la dimensión de N depende del valor de m. Este problema se resuelve por medio de una optimización estructurada, como se verá más adelante.

Las restricciones del diseño son:

- Los rangos naturales de las variables independientes (por ejemplo, el número de bobinas en paralelo puede variar entre uno e infinito).

- Cualquier variable independiente podrá ser fijada por el usuario, lo que se puede formular como restricciones adicionales de rango.

- La reactancia debe tener un coeficiente de autoinducción L comprendido entre ciertos márgenes de error de una L_0 dada (por ejemplo, un 5%).

- La densidad de corriente ∂ en cada una de las bobinas no debe superar un valor máximo ∂_{max}, por propiedades térmicas del esmalte que aisla el conductor.

- El radio interior de la reactancia r está limitado por la elasticidad del conductor, es decir, existe un radio mínimo r_{min} para cada tipo de conductor en función de las dimensiones de su sección {a,b}.

El conjunto de restricciones, aparte de los rangos de las variables independientes, se puede expresar con las siguientes desigualdades:

$$\begin{aligned} L &\leq 1.05\, L_0 \\ L &\geq 0.95\, L_0 \\ \partial_1 &\leq \partial_{max} \\ \cdots &\quad \cdots \quad \cdots \end{aligned}$$

$$\partial_m \leq \partial_{max}$$
$$r \geq r_{min}(a, b)$$

Los requisitos de diseño son los siguientes:

L_0	coeficiente de autoinducción
a,b	dimensiones de la sección del conductor
∂_{max}	densidad de corriente máxima admisible
I_n	intensidad nominal de la línea
$r_{min}(a, b)$	características mecánicas del material conductor

y además:

m, N, r	valores de las variables independientes, si el usuario desea fijar alguna.

Se desea optimizar el diseño de la reactancia minimizando dos características: el **volumen de conductor** utilizado y las **pérdidas totales** por efecto Joule. Son dos **atributos** del diseño conflictivos entre sí, pues para disminuir las pérdidas habrá que aumentar el número de bobinas en paralelo (bajar la densidad de corriente) y eso provoca que aumente el volumen de conductor.

Los atributos del diseño son:

$p = p(m, N, r) =$	$p(X)$	pérdidas totales
$v = v(m, N, r) =$	$v(X)$	volumen de conductor

Las expresiones utilizadas para hallar los coeficientes de inducción mutua y propia de las bobina son desarrollos en serie de convergencia lenta tomados de [Bristol, 45].

Se van a considerar aquí dos **niveles** de detalle en el **modelado**, por lo que podrá descomponerse el proceso de optimización en dos **fases** de diseño. El segundo nivel de modelado será el que ya se ha descrito, con todas las restricciones y variables independientes; el primer nivel de modelado será una simplificación del segundo.

El primer nivel de modelado se basa en suponer que varias bobinas en paralelo son equivalentes a una sola bobina de sección mayor. Las **simplificaciones** correspondientes a esta suposición son las siguientes:

- Todas las bobinas tienen el mismo número de espiras.

- Todas las corrientes de las bobinas son iguales en módulo y fase.
- Se utiliza una fórmula menos precisa que la del segundo nivel de modelado (pero más rápida) para el cálculo del coeficiente de autoinducción L.

Las variables independientes del primer nivel de modelado son:

$$X = \{ m, n, r \}$$

donde:

m	número de bobinas en paralelo
n	número de espiras de las bobinas
r	radio interior de la reactancia

El conjunto de restricciones del primer nivel de modelado se expresa con las siguientes desigualdades:

$$\begin{aligned} L &\leq 1.05\, L_0 \\ L &\geq 0.95\, L_0 \\ \partial &\leq \partial_{max} \\ r &\geq r_{min}(a, b) \end{aligned}$$

A partir de los valores de las variables independientes hay que hallar para cada solución evaluada el valor de las variables dependientes, incluyendo el estado de las restricciones y de los atributos. Esto es lo que se llama **proceso de evaluación.**

El proceso de evaluación del primer nivel de modelado consta de los siguientes pasos:

- El valor de r_{min} se halla al principio del proceso, pues dados {a,b} queda determinado:

$$r_{min} = r_{min}(a, b)$$

- El valor de ∂ debe ser hallado para cada valor de m, pues I_n es constante y {a,b} también:

$$\partial = \frac{I_n}{m\ a\ b}$$

- Los valores de L, p y v deben ser hallados para cada valor de { m, n, r } :

$$L = L(m, n, r)$$ (ver [Bristol, 45])

$$p = (I_n)^2 \rho \frac{n\, 2\pi \left(r + \frac{m\, a}{2} \right)}{m\, a\, b}$$ (siendo ρ la resistividad)

$$v = m\, a\, b\, n\, 2\pi \left(r + \frac{m\, a}{2} \right)$$

En el segundo nivel de modelado, el proceso de evaluación es más complicado:

- Hay que hallar los coeficientes de inducción propia L_{ii} de todas las bobinas, y los de inducción mutua L_{ij} de cada bobina con las demás:

$$M = \begin{bmatrix} L_{11} & L_{12} & \dots & L_{1m} \\ L_{21} & L_{22} & \dots & L_{2m} \\ \dots & \dots & \dots & \dots \\ L_{m1} & L_{m2} & \dots & L_{mm} \end{bmatrix}$$

- Hay que hallar la distribución de corrientes, resolviendo el circuito:

$$\begin{bmatrix} V \\ \dots \\ V \end{bmatrix} = \begin{bmatrix} L_{11} & \dots & L_{1m} \\ \dots & \dots & \dots \\ L_{m1} & \dots & L_{mm} \end{bmatrix} \begin{bmatrix} I_1 \\ \dots \\ I_m \end{bmatrix}$$

$$I_n = \sum_{i=1}^{m} I_i$$

- Se hallará entonces el valor de la L conjunta:

$$L = \frac{V}{I_n}$$

- Se hallarán las densidades de corriente ∂_i:

$$\partial_i = \frac{I_i}{a\, b}$$

- Se hallarán las pérdidas totales p :

$$p = \sum_{i=1}^{m} (I_i)^2 \rho \frac{n_i 2 \pi r_i}{a b}$$

A lo largo de este capítulo se irá formalizando y modelando un proceso de diseño general, tomando el caso de este apartado como ejemplo particular. Ya se verá cómo se puede utilizar el primer nivel de modelado para obtener buenos puntos iniciales para el segundo, y, más adelante, cómo se puede solucionar una optimización en un espacio de variables independientes de dimensión variable (es el caso del vector N de este ejemplo).

3.3 - Formalización del Problema General.

En la introducción de la tesis (Capítulo 1) se mencionó la existencia de dos grandes modelos alternativos para el problema del diseño: el modelo lingüístico y el matemático. En esta tesis se escoge decididamente el modelo matemático por diversas razones:

- La optimización se considera un problema fundamentalmente matemático, por lo que una herramienta competitiva debe garantizar la optimalidad de las soluciones ofrecidas siguiendo criterios matemáticos.

- Los ordenadores trabajan internamente con variables y funciones (memories and processors, según [McFarland et al, 90]), a lo que un modelo matemático se adapta de forma natural. Nótese que en cuanto a cálculos matemáticos el hombre no puede ni soñar en competir con las máquinas, lo que sugiere modelar matemáticamente el conocimiento para que los ordenadores sean más eficaces que el hombre.

- En la representación de un problema podrá utilizarse un modelo lingüístico, pero en su resolución se usa siempre un algoritmo, lo que implica desarrollar el modelo matemático (variables y funciones) correspondiente a dicho algoritmo. Hay problemas en los que por su propia naturaleza hacen falta ambos modelos, como por ejemplo la interpretación y construcción de lenguajes; sin embargo, la mayoría de los problemas de diseño se

pueden abordar directamente con un modelo matemático, facilitando el desarrollo de algoritmos eficaces.

- Los ingenieros están más habituados a trabajar con modelos matemáticos que con modelos lingüísticos.

Nota bibliográfica:
En [Grossman, 90] se indica que cualquier sistema de reglas lógicas de implicación puede expresarse usando un modelo matemático como un sistema de restricciones lineales afectando a un conjunto de variables booleanas (discretas 0-1).
Ya se ha comentado que en el problema de diseño de circuitos lógicos integrados es apropiado el uso de modelos lingüísticos para las primeras fases de diseño, pues el comportamiento del circuito se especifica mediante código escrito en un lenguaje de programación dado.

En este apartado se formaliza paso a paso el problema general de diseño desde un punto de vista puramente matemático, concretándose de forma ordenada el significado y relaciones mutuas de los elementos del problema de diseño identificados en el apartado 3.1.

El diseño de un aparato o sistema quedará totalmente caracterizado por el valor de un conjunto de variables de diseño que aparecen al modelar dicho sistema. Sea XX el vector de **nn variables** que intervienen en el problema de optimización de diseño:

$$XX = \{ x_1, x_2, ..., x_{nn} \} \in \Re^{nn}$$

En el ejemplo del apartado 3.2 las variables de diseño son:

- El número de bobinas en paralelo.
- El número de espiras de cada bobina.
- El radio interior de la reactancia.
- La altura y el radio de cada bobina.
- Los coeficientes de autoinducción de cada bobina y los de inducción mutua con todas las bobinas con las demás.
- La corriente y la densidad de corriente en cada bobina.
- El volumen de conductor y las pérdidas totales en la reactancia.
- etc.

Sea G(XX) un conjunto de **p relaciones de ligadura** no redundantes:

$$G(XX) = \begin{cases} g_1(XX) = 0 \\ g_2(XX) = 0 \\ \quad \ldots \quad \ldots \\ g_p(XX) = 0 \end{cases}$$

Se selecciona un subconjunto X de n variables del conjunto XX, por ejemplo las n primeras componentes de XX, que serán las **variables independientes**:

$$X = \{ x_1, x_2, ..., x_n \} \quad \in \quad \Re^n$$

donde

$$n = nn - p$$

A partir del valor de las variables independientes se debe poder hallar el valor de todas las variables de diseño, ya sea de forma explícita o implícita. En el caso ejemplo del apartado anterior las **variables independientes** que se consideran son:

$$X = \{ m, n_1, n_2, ..., n_m, r \} = \{ m, N, r \}$$

donde:

m	número de bobinas en paralelo
$N = \{ n_1, n_2, .., n_m \}$	número de espiras de cada bobina
r	radio interior de la reactancia

Las ecuaciones de ligadura son las que permiten hallar el resto de variables de diseño a partir de las variables independientes. En el ejemplo del apartado 3.2 estas ecuaciones son las que componen el proceso de evaluación.

La variable vectorial X de variables independientes contiene un conjunto mínimo (no redundante) de información sobre el sistema a diseñar. Dicha variable vectorial define un **espacio vectorial** que en general será de dimensión variable (como se puede ver en el caso ejemplo) asociado a una clase lógica: la de posibles diseños o puntos del **espacio de diseños**. Cada individuo de esta clase es un posible diseño desde un punto de vista matemático, lo cual no quiere decir que todos sean diseños físicamente posibles ni que sean factibles.

En el ejemplo de la reactancia queda muy claro por qué el espacio de variables de diseño y el de variables independientes puede ser de dimensión variable. La dimensión de XX y la dimensión de X dependen de una componente de ambos: el número de bobinas m.

No todos los puntos del espacio de variables independientes corresponden a diseños posibles. La región de diseños posibles está limitada inicialmente por los rangos de variación de dichas variables (que son restricciones propias del modelo), dando lugar a una **región de búsqueda** inicial fuera de la cual no tiene sentido evaluar soluciones. Una región de búsqueda se define por medio de una serie de desigualdades (o inecuaciones) del tipo:

$$\begin{aligned} xmin_1 \leq \quad & x_1 \quad \leq xmax_1 \\ xmin_2 \leq \quad & x_2 \quad \leq xmax_2 \\ \ldots \quad & \ldots \quad \ldots \\ xmin_n \leq \quad & x_n \quad \leq xmax_n \end{aligned}$$

La figura 3.2 muestra la forma de una región de búsqueda en un espacio de variables independientes de dos dimensiones.

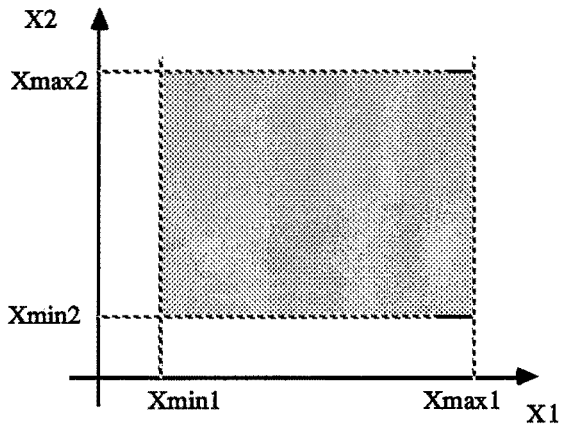


Figura 3.2. Región de Búsqueda

En el caso ejemplo, la región inicial de búsqueda viene definida por las siguientes restricciones en los rangos de las variables independientes:

$$\begin{aligned} 0 < \quad & m \quad < \infty \\ 0 < \quad & n_i \quad < \infty \\ \ldots \quad & \ldots \quad \ldots \\ 0 < \quad & n_m \quad < \infty \\ 0 < \quad & r \quad < \infty \end{aligned}$$

A estas restricciones de rangos hay que añadir las restricciones propias del problema de diseño. Estas restricciones podrán ser impuestas por el cliente, por normas o por limitaciones físicas de materiales y subsistemas. Sea **F(XX) un conjunto de r restricciones de diseño:**

$$F(XX) = \begin{cases} f_1(XX) \leq 0 \\ f_2(XX) \leq 0 \\ \quad \ldots \quad \ldots \\ f_r(XX) \leq 0 \end{cases}$$

Entre las restricciones establecidas en el caso ejemplo del apartado 3.2, el coeficiente de autoinducción podría ser impuesto por el cliente, la densidad de corriente máxima por normas y el radio mínimo por limitaciones físicas de materiales. Estas tres restricciones se pueden expresar mediante las siguientes inecuaciones:

$$\begin{array}{lcl} L & \geq & 0.95\ L_0 \\ L & \leq & 1.05\ L_0 \\ \partial_1 & \leq & \partial_{max} \\ \ldots & \ldots & \ldots \\ \partial_m & \leq & \partial_{max} \\ r & \geq & r_{min} \end{array}$$

La región del espacio de variables independientes que cumple todas las restricciones (incluidas las de rango de variables) se llama **región factible**. En la figura 3.3 se muestra en un espacio de dos dimensiones cómo la región inicial de búsqueda queda limitada por restricciones adicionales, dando lugar a la región factible.

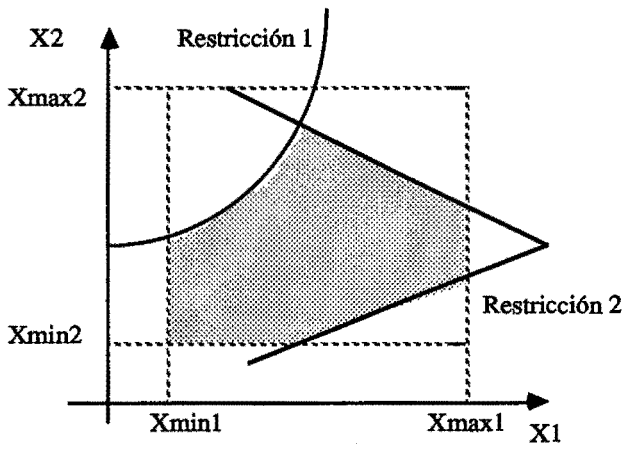


Figura 3.3. Región Factible

Para realizar una optimización de diseño sistemática no basta con poder comprobar si un diseño es o no factible. Es necesario definir criterios numéricos para poder establecer el criterio de "mejor" y por tanto también el de "óptimo". Entre dos diseños no factibles habrá que considerar mejor al diseño que esté más cerca de ser factible, esto es, que viole en menor medida el conjunto de restricciones. Tendrá que definirse por tanto una medida de la violación de cada restricción y una ponderación o importancia relativa de las restricciones entre sí.

Una manera de cuantificar la violación de las restricciones consiste en definir el **vector de estado de las restricciones E**. Cada componente de este vector está asociada a una restricción, y valdrá cero si la restricción está justo al límite, positiva si está violada y negativa si no lo está. El valor de la componente correspondiente tomará un valor proporcional a la "distancia" entre el punto y la frontera de la restricción:

$$E = E(X) = \{ e_1(X), e_2(X), \ldots \}$$

El vector E define el **espacio de restricciones**, en el cual toda solución perteneciente a la región de búsqueda debe poder situarse. En la figura 3.4 se muestra cómo a cada punto de la región de búsqueda le corresponde un punto del espacio de restricciones por medio de la función o proceso de **evaluación**.

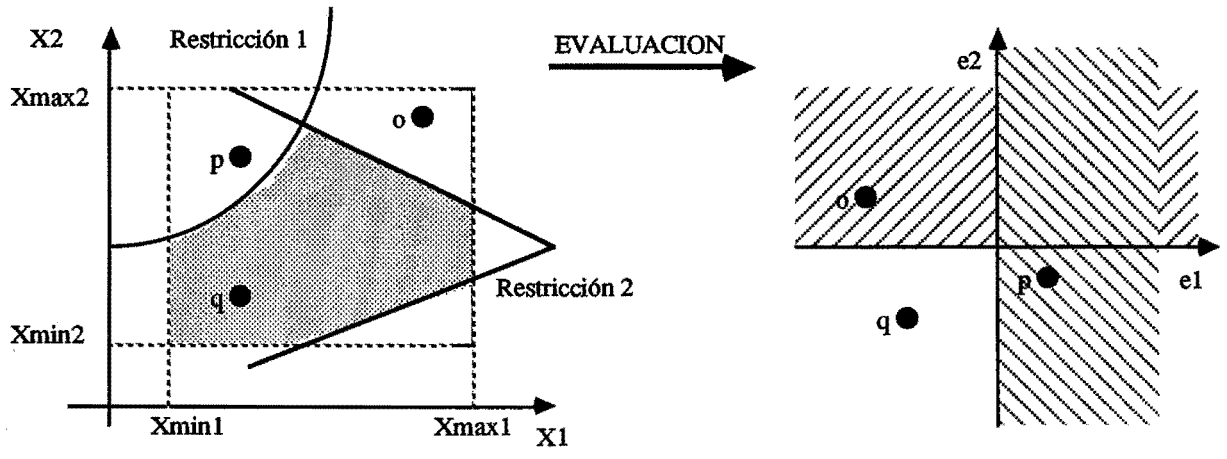


Figura 3.4. Espacio de Restricciones

Por ejemplo, en el caso de la reactancia, el vector E de estado de las restricciones podría tener tres componentes, o bien ser de dimensión variable (si se desea tratar por separado cada bobina). También se podría considerar por separado el caso de no cumplir la restricción de L por exceso y el de no cumplirla por defecto; en general depende de la información que se necesite en la estrategia de optimización usada. Una posibilidad sería definir E como:

$$E = \{ e_L(X), e_\partial(X), e_r(X) \}$$

donde por ejemplo e_r se podría hallar con la siguiente expresión:

$$e_r(r) = 100 \frac{r_{min} - r}{r_{min}}$$

Mediante el vector de estado de las restricciones se pueden comparar diseños no factibles. Pero en el proceso de optimización habrá también que comparar entre sí diseños factibles según los atributos del diseño. Los **atributos** de un problema de diseño son variables escalares cuyos valores se utilizan como medidas de la bondad de cada solución particular. El conjunto de atributos se puede agrupar en una sola variable vectorial A :

$$A = A(X) = \{ a_1(X), a_2(X), \ldots \}$$

El vector A define el **espacio de atributos**, en el cual toda solución perteneciente a la región de búsqueda debe poder situarse. En la figura 3.5 se muestra cómo a cada punto de la región de búsqueda le corresponde un punto del espacio de atributos por medio de la función o proceso de **evaluación**.

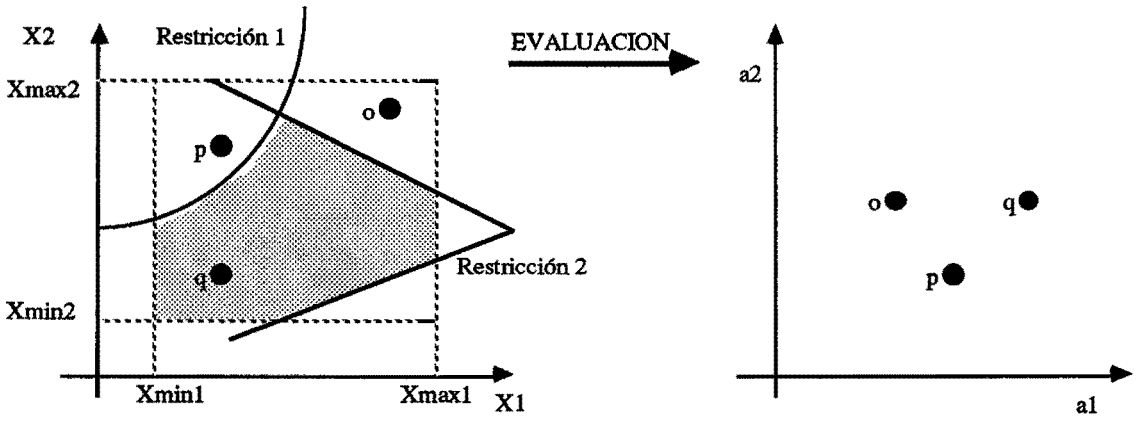


Figura 3.5. Espacio de Atributos

Los atributos del ejemplo de estudio son:

$$A = \{ p(X), v(X) \}$$

$p = p(m, N, r) =$	$p(X)$	pérdidas totales (ver apartado 3.2)
$v = v(m, N, r) =$	$v(X)$	volumen de conductor (ver apartado 3.2)

Hay que considerar también la comparación entre un diseño factible y otro que no lo es. Debe tenerse en cuenta que un diseño provisional ligeramente no admisible pero con atributos buenos puede ser mejor que otro admisible con atributos malos en cuanto al futuro de la optimización.

Para comparar puntos admisibles con otros que no lo son se propone aquí adoptar la solución aplicada en trabajos como [Pérez-Arriaga, 78], [Cuadra et al, 85], [IIT, 87] y [García et al, 87]. Consiste en tratar el estado de las restricciones como un atributo ficticio que se desea minimizar, ponderando la violación de cada restricción con un factor proporcional a su importancia relativa.

Es frecuente que en un problema de optimización haya que considerar más de un atributo para juzgar cada solución y comparar distintas soluciones entre sí. También es frecuente que algunos atributos sean mutuamente conflictivos; se dice que dos **atributos** son **mutuamente conflictivos** cuando al intentar mejorar uno de ellos el otro tiende a empeorar [Schweppe & Merril, 86]. Cuando en un proceso de optimización se consideran dos o más atributos, sean o no mutuamente conflictivos, se dice que dicho proceso es una **optimización multiatributo.**

En el ejemplo de la reactancia los atributos considerados son mutuamente conflictivos. Al intentar minimizar las pérdidas se tiende a aumentar la sección de conductor, aumentando su volumen; al intentar minimizar el volumen, aumenta la densidad de corriente y por tanto las pérdidas aumentan. Esto no impide que algún cambio en el diseño pueda mejorar o empeorar ambos atributos simultáneamente; la conflictividad suele aparecer en los límites de la zona factible.

Además de optimización multiatributo, también se utiliza el término **optimización multicriterio.** Por ejemplo, en [Borrel, 87] se llama optimización multicriterio a la que trata de optimizar una función vectorial en vez de una función escalar.

El resultado teórico de una optimización monoatributo es un sólo punto: el óptimo absoluto. El resultado teórico de una optimización multiatributo es una hipersuperficie en el espacio de atributos, representada en la práctica por un conjunto de puntos tales que todos presentan cualidades y defectos frente a los demás, necesitándose una solución de compromiso para seleccionar uno de ellos.

En los problemas de optimización es muy habitual reducir todos los criterios por los cuales se evalúa una solución (tanto restricciones como atributos) por medio del valor de una única función escalar cuyo valor se trata de optimizar. Esta función se suele llamar función objetivo.

Una **función objetivo** es una función escalar de variable vectorial cuyos argumentos son el vector de atributos A(X) y el vector de estado de las restricciones E(X). Es frecuente que, para definir esta función, el estado de restricciones se formule como un conjunto de atributos ficticios. El problema se plantea como la optimización de una función objetivo, para lo cual es necesario contar con un **punto inicial** y una estrategia de búsqueda.

La **estrategia de búsqueda** es el algoritmo responsable de emular dos de las tareas habitualmente realizadas por el ingeniero de diseño: la primera consiste en realizar cambios en un diseño conociendo el estado de las restricciones y de los atributos de los diseños precedentes; la segunda consiste en decidir cuándo se ha alcanzado un óptimo local, si se sigue una búsqueda de tipo local, o cuándo se ha explorado suficientemente el árbol de posibilidades, si se sigue una búsqueda en árbol.

Se define aquí un **óptimo local**, respecto a una función objetivo y una estrategia de búsqueda local dadas, como un punto del espacio de variables independientes tal que partiendo de dicho punto y siguiendo dicha estrategia no se puede encontrar otro punto en el que la función objetivo tenga un valor mejor. Esta definición es válida para cualquier tipo de espacio de búsqueda (variables continuas o discretas, dimensión variable o constante), para cualquier estrategia de búsqueda local y cualquier función objetivo (continua, discreta, derivable o no, redefinida, etc.).

En rigor debe además señalarse que la condición de óptimo local depende también del tamaño de los incrementos mínimos de las variables independientes, especialmente en las variables llamadas continuas. Esto está relacionado con el concepto de "tamaño" de una variable, descrito ampliamente en el apartado 5.1 - Criterios para la Clasificación de las Variables Independientes.

Si se desea obtener una sola solución al problema, bastará definir una función objetivo. Sin embargo, si se desea obtener un conjunto de soluciones representativas de la hipersuperficie ya mencionada (optimización multiatributo) se puede definir un conjunto de funciones objetivo distintas, como se hace en [IIT, 87], obteniendo así un conjunto de óptimos locales presumiblemente distintos.

El problema de optimizar un diseño quedaría así reducido a maximizar o minimizar un conjunto **FO(X)** de **nf** funciones objetivo (en este trabajo se considera que toda función objetivo debe ser minimizada); cada función objetivo se minimiza independientemente y representa una valoración distinta de las importancias relativas de atributos y restricciones.

La optimización multiatributo se expone con detalle en el Capítulo 6, pero se puede anticipar un ejemplo tomado del caso de estudio. Para optimizar el diseño de la reactancia con respecto a los dos atributos y además tratar las restricciones como un atributo más, se puede definir un conjunto de funciones objetivo de la forma:

$$FO(X) = \begin{cases} fo_1(X) \\ fo_2(X) \\ \dots \\ fo_{nf}(X) \end{cases}$$

$$fo_i(X) = wp_i\ p(X) + wv_i\ v(X) + wL_i\ e_L(X) + w\partial_i\ e_\partial(X) + wr_i\ e_r(X)$$

donde { wp_i , wv_i , wL_i , $w\partial_i$, wr_i } es el vector de los respectivos pesos de cada sumando de la función objetivo, y donde wL_i, $w\partial_i$ y wr_i pueden ser cero si la restricción correspondiente no está violada (e(X) negativas). Los pesos son distintos para cada $fo_i(X)$, expresando cada conjunto de pesos un punto de vista distinto al optimizar.

Por último, los **requisitos** del diseño son las variables de entrada de la función "resolver el problema de diseño". Estas variables pueden afectar a restricciones, atributos, ligaduras, etc., y cambiarán de valor de un caso a otro.

En el ejemplo de la reactancia los requisitos del diseño son:

L_0	coeficiente de autoinducción
a,b	dimensiones de la sección del conductor

∂_{max}	densidad de corriente máxima admisible
I_n	intensidad nominal de la línea
$r_{min}(a, b)$	características mecánicas del material conductor

y además:

m, N, r	valores de las variables independientes, si el usuario desea fijar alguna.

3.4 - Enfoque Propuesto: Optimización Estructurada Multiatributo.

En este apartado se describe cómo se puede abordar el desarrollo de herramientas informáticas de optimización de diseño. Estas herramientas no serían de aplicación universal, sino que cada una estaría concebida a medida de las características de cada problema particular. El problema de diseño más complejo que se pretende resolver aquí reune las siguientes dificultades:

- Hay un número elevado de variables independientes (o grados de libertad).

- La dimensión del vector de variables independientes es variable, por ser función del valor de algunas de sus componentes.

- Las variables independientes son de naturaleza matemática distinta (discretas y continuas mezcladas) y también de desigual importancia relativa en cuanto a su impacto en las características del diseño.

- Hay un número elevado de restricciones de diseño.

- Es un problema de optimización realmente multiatributo, es decir, no es útil reducir todos los atributos a una sola función objetivo.

- Los procesos de cálculo (ecuaciones de ligadura) necesarios para evaluar las soluciones son complicados y lentos.

Para afrontar la gran complejidad de un problema de diseño con estas características se propone la Optimización Estructurada Multiatributo. El término "estructurada" hace referencia a la descomposición de un problema complejo de optimización en un sistema estructurado de subproblemas más simples.

3.4.1 - Adquisición del Conocimiento y Modelado del Problema.

Esta es una etapa fundamental en el desarrollo de un proyecto de Ingeniería del Conocimiento. La persona que realmente conoce el problema (experto) no tiene habitualmente tiempo ni conocimientos para construir una herramienta informática, y la persona capaz de diseñar y construir buenas herramientas informáticas (ingeniero del conocimiento) no suele conocer el problema.

En el apartado 2.3 se propuso un método sistemático para la extracción del conocimiento, apoyado en un sistema gráfico de representación. Se va a especificar ahora qué conocimiento se debe obtener, y en qué orden se sugiere obtenerlo, para el desarrollo sistemático de una herramienta informática de optimización de diseño.

Es en general recomendable comenzar modelando el problema de diseño según el caso más general y con el máximo nivel de detalle posible, y representar formalmente dicho modelo. Primero, porque resulta más sencillo simplificar después los modelos que complicarlos; segundo, porque se tienen así controlados los errores cometidos, pues los resultados obtenidos con los modelos simplificados se podrán comparar con los de los modelos más precisos.

El primer paso a seguir es la identificación exhaustiva de todos los **atributos** y todas las **restricciones** posibles del problema de diseño. Es frecuente que algunas características del diseño puedan ser consideradas tanto atributos como restricciones, según los deseos del usuario; en ese caso, se deben incluir en ambos grupos, buscando siempre el caso más general.

Una vez hecho esto ya se tiene una lista de todas las variables que es necesario conocer para evaluar cada solución. Se pueden ahora identificar todos los **procesos (o funciones) de cálculo** necesarios para obtener esas variables, es decir, para conocer el valor de los atributos y el estado de las restricciones.

El siguiente paso es muy importante y debe llevarse a cabo con un conocimiento preciso del problema. Consiste en intentar unir todos estos procesos de cálculo en un sólo proceso, llamado aquí **proceso de evaluación**. El resultado de este paso es múltiple:

i) Intentar enlazar procesos de cálculo diversos revela la necesidad de **procesos de cálculo auxiliares** o intermedios, y también la existencia de subprocesos **comunes** a varios procesos.

ii) Cada proceso de cálculo tiene sus propias variables de entrada; al intentar enlazar todos los procesos, se descubre la existencia de **variables comunes**, y además se identifica cuál es el conjunto de **variables independientes** más adecuado según el orden lógico de los cálculos.

iii) Se pueden detectar **iteraciones** entre procesos, por ser las entradas de uno salidas de otro y viceversa, es decir, por formar bucles de cálculo. El caso más general es que un grupo de procesos forme un auténtico sistema de macroecuaciones (procesos de cálculo) cuya **convergencia** se garantice con algoritmos de control.

iv) Se detectará la existencia de **procesos** de cálculo **condicionados** al valor de ciertas variables. Por ejemplo, el proceso de análisis térmico de un aparato dependerá del tipo de sistema de refrigeración empleado, y el análisis de la aplicación de cada sistema de refrigeración exige un proceso de cálculo distinto.

v) Se puede obtener una lista exhaustiva de todos los **requisitos** relacionados con el proceso de evaluación. Se verá luego que hay otro grupo de requisitos relacionados con los procesos de búsqueda que sólo pueden especificarse al desarrollar éstos.

Una vez definido, el proceso de evaluación se puede estudiar con detalle. En particular, se debe estimar el **tiempo de proceso** necesario en cada evaluación, y se deben identificar cuáles son los **procesos más lentos**. Para estos procesos hay que estudiar modelos alternativos más rápidos y los errores que llevan asociados. El Capítulo 7 de esta tesis está dedicado íntegramente a la sustitución de procesos lentos por funciones de interpolación (ábacos), como alternativa a la utilización de modelos simplificados.

La existencia de varios niveles de detalle, de precisión y de velocidad en el proceso de evaluación permite la descomposición del problema en **fases de diseño**. Cada fase de diseño

tendrá su propio proceso de evaluación, lo que implica que también puede tener sus propias variables independientes, atributos y restricciones.

Cada fase de diseño es un problema de optimización completo y distinto de las demás fases. Si es suficientemente complejo, cada problema de optimización se puede **descomponer** en un sistema estructurado de **subproblemas** de cálculo y de optimización, como ya se verá más adelante.

La descomposición en subproblemas de optimización exige estudiar a fondo las características de las variables seleccionadas como independientes. Hay especialmente dos características muy útiles que se describen en el apartado 5.1: el **tamaño** y el **comportamiento local** de cada variable independiente.

Al incorporar los procedimientos de optimización a los de evaluación aparecerán nuevos **requisitos**, relacionados con las estrategias de búsqueda empleadas (por ejemplo, los incrementos de las variables, los pesos relativos de las restricciones, etc.).

Una vez que todos los requisitos han sido identificados, ya se conocen las variables de entrada. Se debe ahora determinar qué variables son la salida del problema, aparte de las soluciones obtenidas. Todas las variables que aportan alguna información interesante sobre las soluciones y sobre el proceso de resolución en sí se pueden incluir en **informes** para el usuario. Normalmente los informes serán muy extensos en las etapas de desarrollo y prueba de la herramienta, y más concisos cuando ya se tenga confianza en sus resultados.

Los interfaces con el usuario son, como se ha podido apreciar, los últimos en definirse. Sin embargo es conveniente que sean los primeros en implantarse, tanto los de entrada de datos por ficheros como los de entrada por teclado. Esto ahorra mucho tiempo durante el desarrollo de la herramienta, y hace más agradable tanto la programación como la búsqueda de errores.

3.4.2 - Descomposición del Problema en Fases de Diseño.

En el apartado 3.1 se vio cómo en la literatura técnica se suele descomponer el proceso de diseño en varias fases. Aunque en general las descomposiciones propuestas no resultan útiles, hay dos fases que merece la pena destacar por ser las únicas interactivas: la primera, llamada aquí "análisis de consistencia y prediseño" se encarga de los datos de entrada; la segunda, el

"interfaz final", elabora los informes que necesita el usuario sobre las soluciones y su obtención.

La figura 3.6 muestra el proceso general de diseño descompuesto en fases. La estructura que presenta es la de una cadena de fases de diseño tendida entre dos procesos especiales, que son los únicos interactivos. En la figura todas las fases intermedias están representadas por dos, pero en un problema más complejo podría haber más fases de diseño.

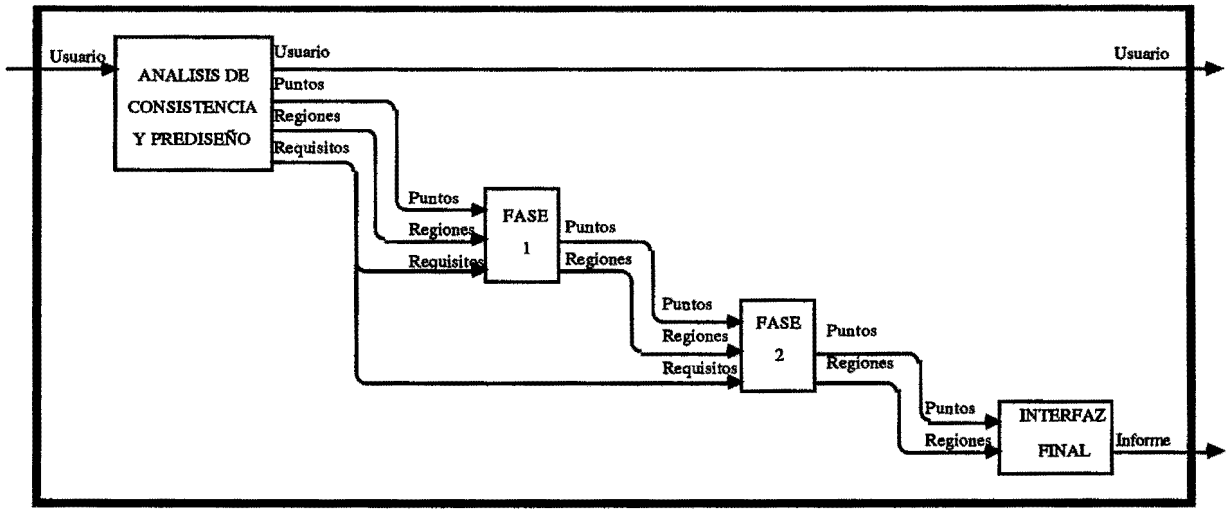


Figura 3.6. Estructura General del Proceso de Diseño

El proceso de **análisis de consistencia y prediseño** podría considerarse en teoría una fase de diseño más; pero en realidad este proceso tiene una estructura y unas funciones muy distintas porque debe comunicarse de forma interactiva con el usuario y con la base de datos del problema.

En esta base de datos están definidos todos los espacios vectoriales de los elementos del proceso: el espacio total de variables independientes, el de restricciones, el de requisitos, el de atributos, y los espacios de funciones objetivo y estrategias de búsqueda. También en la base de datos se puede tener almacenada la información de un sistema de memoria capaz de almacenar soluciones de problemas de diseño anteriores, interesantes para generar buenos puntos iniciales: esto se denomina "sistema de memoria selectiva" (ver apartado 7.4).

El análisis de consistencia será capaz de comprobar la viabilidad del diseño de una forma aproximada, y puede incluir aplicaciones de normativas tanto como cálculos sencillos. El proceso será interactivo, y al final del mismo quedará establecido por completo el problema de

diseño en todas sus fases: grados de libertad (el usuario podrá siempre fijar algunas de las variables independientes), atributos a considerar, funciones objetivo y estrategias de búsqueda. Todo esto se puede resumir en que tras este proceso se debe obtener un conjunto de requisitos coherente y completo.

Es esta fase la que parece más indicada para aplicar las técnicas clásicas de la Ingeniería del Conocimiento (no englobadas normalmente en los textos de optimización matemática), incluyendo el razonamiento cualitativo, el razonamiento funcional (y propagación de restricciones), las memorias asociativas, los sistemas de reglas de producción, etc.

El **interfaz final** es el proceso encargado de seleccionar y presentar los resultados finales al usuario, y si éste lo desea también un informe sobre la marcha del proceso de optimización. Esto será especialmente útil en la etapa de desarrollo de la herramienta de diseño, para estudiar el comportamiento de la misma. Por supuesto que el grado de transparencia y el de flexibilidad de la herramienta deben ser grandes sobre todo en la etapa de desarrollo; posteriormente no será necesario tener acceso a detalles como "¿cuál estrategia de búsqueda se usa?" ni tampoco obtener informes exhaustivos de los procesos intermedios.

Nota bibliográfica:
La fase de análisis de consistencia y prediseño aparece con distintos nombres en muchas publicaciones: requirements en [Brown & Chandrasekaran, 85], [Han et al, 87]; predesign en [Rehak et al, 85]; preliminary design en [Shukla, 88]; interviews en [Bowen, 85]; specification en [Dietterich & Ullman, 87]; intelligent interface en [Iwata et al, 87]. Parte de la misión de esta fase es la que se describe en [Veinott, 72] como "establish performance requirements and evaluation criteria". También se describe como "determine design specifications" y "conceptual (preliminary) design" en [Zhang, 89].
El interfaz final aparece con distintos nombres en las publicaciones, como por ejemplo "reports" en [Bowen, 85].

El objetivo de cualquier **fase del diseño** (incluyendo la de prediseño y análisis de consistencia) consiste en producir un conjunto de **regiones de búsqueda**, definidas por un conjunto de variables independientes y por los rangos de variación de dichas variables. Para cada región de búsqueda estará definido un conjunto de puntos, pertenecientes a la región, tales que puedan ser utilizados como buenos **puntos iniciales** para las búsquedas de la siguiente fase.

El objetivo de la siguiente fase es producir otro conjunto de regiones de búsqueda y, dentro de cada una de ellas, un conjunto de óptimos locales para poder comenzar la fase posterior, y así encadenar todo el proceso de diseño. Los puntos de interés obtenidos por la última fase de diseño son las soluciones del problema.

El hecho de hablar de regiones y puntos en plural se debe a dos circunstancias totalmente independientes, pero que obligan por separado a conservar más de un punto:

i) El carácter multiatributo de la optimización, que hace que no haya un óptimo absoluto, sino una hipersuperficie de puntos óptimos.

ii) La incertidumbre en la evaluación por la utilización de modelos simplificados, que obliga a ser conservador y a no despreciar soluciones alternativas cuya evaluación sea parecida.

En el caso ejemplo de la reactancia se puede descomponer la optimización en dos fases, según los dos modelos del problema descritos en el apartado 3.2: recuérdese que la primera fase tiene menos variables independientes y que el proceso de evaluación es mucho más rápido.

A continuación se va a utilizar de nuevo la reactancia para mostrar ejemplos de regiones de búsqueda y puntos iniciales.

Los resultados de la primera fase no son muy precisos, por despreciar la dispersión magnética entre bobinas en paralelo y el reparto desigual de intensidades. Pero los resultados aportan una información muy valiosa para la segunda fase, porque se sabe el orden de magnitud de las soluciones. Por ejemplo, los valores alternativos de la variable "número de bobinas en paralelo" (m) podrían quedar reducidos a dos, por haber dos atributos conflictivos:

- El primero, el valor de m mínimo para cumplir la restricción térmica (densidad de corriente), con objeto de minimizar el volumen de conductor v.
- El segundo, un valor de m mayor, para optimizar las pérdidas p dejando el volumen de conductor en un valor razonable.

Esto se puede expresar en el algoritmo de dos formas distintas: como una única región de búsqueda con dos valores posibles para la variable m, o bien como dos regiones de búsqueda en las que la variable m es constante (pero tiene un valor distinto en cada región).

Otra variable que se puede acotar fácilmente tras la primera fase es el número de espiras de cada bobina, pues se puede estimar el error cometido en la primera fase en el coeficiente de autoinducción total; el valor inicial del número de espiras de todas las bobinas puede ser también el valor óptimo obtenido en la primera fase.

La segunda fase tendría de este modo que explorar unas regiones de búsqueda más limitadas, acortando el proceso de búsqueda. Además, la primera fase proporcionaría buenos **puntos iniciales** (al menos uno en cada región) para la segunda fase, dando un número de espiras n y un radio interior r razonables para obtener el coeficiente de autoinducción deseado, y un número de bobinas razonable según cada criterio de optimización.

3.4.3 - Descomposición de una Fase de Diseño en Subproblemas.

Ya se ha visto que cada fase de diseño es un problema de optimización multiatributo distinto, con su propio nivel de detalle en el modelado del problema. Esto implica tener su propio proceso de evaluación y sus propias estrategias de búsqueda, con su espacio de variables independientes, sus regiones de búsqueda, puntos iniciales, restricciones y atributos.

La figura 3.7 muestra la organización interna de una fase de diseño genérica. En ella aparecen las funciones o procesos fundamentales y sus intercambios de datos.

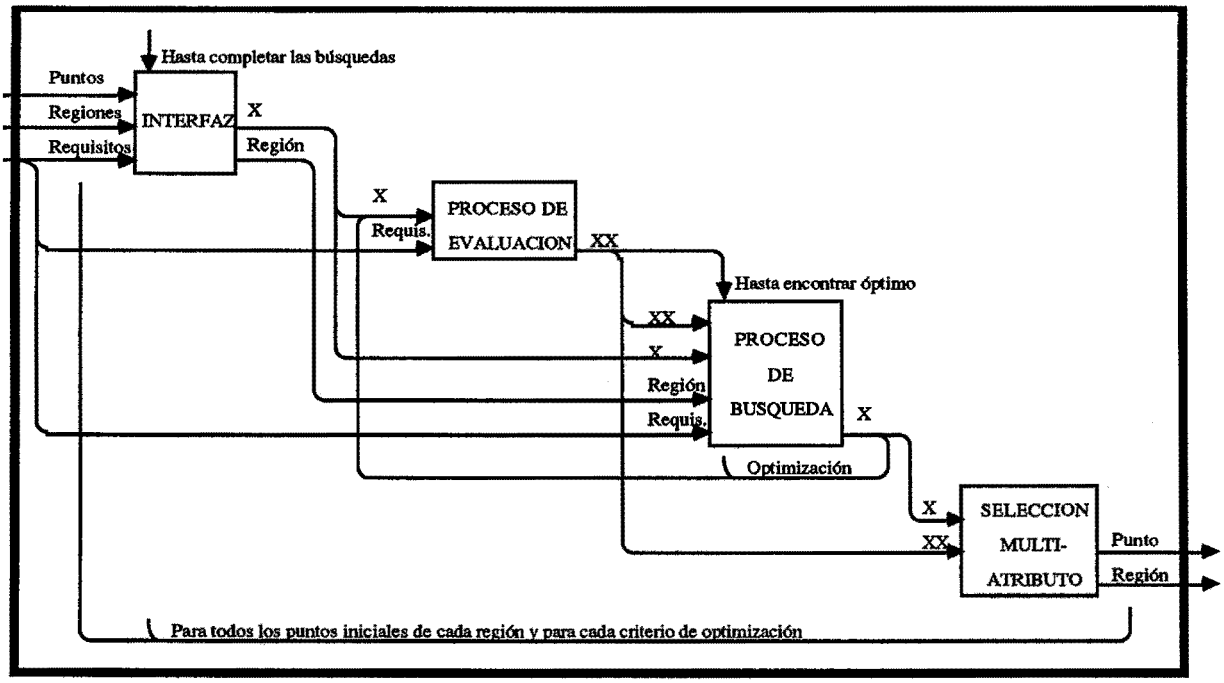


Figura 3.7. Estructura Interna de una Fase de Diseño

El **interfaz** es el proceso encargado de trasladar los puntos y las regiones de búsqueda de la fase anterior al espacio de variables independientes de la fase en curso. Esta tarea puede ser

en muchos casos trivial, pero no en todos, pues es muy frecuente que el cambio de espacio implique también un cambio de dimensión.

Por ejemplo, en el caso de la reactancia hay un traslado de espacio entre los dos niveles de detalle del modelado de la reactancia, o sea, entre las dos fases de diseño planteadas. Supóngase que en la primera fase se producen dos óptimos locales:

- Optimo 1) $X = \{m, n, r\} = \{ 2, 10, 1.0 \}$
- Optimo 2) $X = \{m, n, r\} = \{ 3, 11, 0.9 \}$

Supóngase que en la segunda fase se va a mantener constante el número de bobinas, con lo que m ya no será una variable independiente, sino una constante. El interfaz de la segunda fase debe producir dos puntos iniciales del espacio de variables independientes de la segunda fase, cada uno correspondiente a un óptimo distinto de la fase anterior. El óptimo 1 da un punto inicial en un espacio de búsqueda de dimensión 3, mientras que el óptimo 2 da un punto de un espacio de búsqueda de dimensión 4:

- Punto inicial 1) $X = \{n_1, n_2, r\} = \{ 10, 10, 1.0 \}$

- Punto inicial 2) $X = \{n_1, n_2, n_3, r\} = \{ 11, 11, 11, 0.9 \}$

La **selección multiatributo** es el proceso encargado de filtrar los puntos obtenidos al optimizar en distintas regiones, con distintos puntos iniciales y con distintos criterios de optimización. Sólo los puntos que parezcan pertenecer a la hipersuperficie de soluciones óptimas pasarán como puntos de interés a la siguiente fase de diseño, teniendo en cuenta la incertidumbre en la evaluación por emplear modelos simplificados. Los criterios de selección multiatributo se tratarán en el Capítulo 6.

El **proceso de evaluación** es la función que relaciona las variables dependientes con las independientes, y se resolverá con un algoritmo de cálculo que puede ser muy complejo. En el grupo de las variables dependientes se encuentran los atributos, el estado de las restricciones, los parámetros de sensibilidad de las funciones (como por ejemplo los gradientes) y el valor de las funciones objetivo. Todo esto se engloba en la figura 3.7 dentro de la variable vectorial de variables de diseño (XX).

Pueden surgir dos dificultades en el proceso de evaluación: la primera consiste en que las variables dependientes no sean directamente despejables de las ecuaciones de ligadura,

exigiendo la resolución de grandes sistemas de ecuaciones no lineales que pueden presentar problemas de convergencia. La segunda es la posible lentitud del proceso de cálculo, que se hace patente cuando se ha de evaluar un gran número de puntos.

El problema de la **convergencia** de los cálculos se agudiza cuando se desea permitir al usuario determinar las variables independientes con total flexibilidad. Este tema se trata en trabajos muy interesantes, como pueden ser [Elías, 85], [Kolb, 86a], [Kolb, 86b] y [Lajoie, 86], y consiste en resolver las relaciones de ligadura propias del modelado a partir de cualquier conjunto de variables independientes, usando algoritmos guiados por heurísticos. En el Capítulo 8 (futuros desarrollos) se comenta este problema y se esboza un método original todavía no aplicado para generalizar la inversión numérica de funciones.

Sin embargo, la experiencia parece demostrar que el cambio en la selección de las variables independientes en tiempo de proceso no es necesario ni conveniente:

- No es necesario porque si realmente se desea controlar el valor de una variable que por la naturaleza del diseño es dependiente, se puede definir como una restricción o un atributo; el problema se reduce entonces a una selección adecuada de las variables independientes, al menos en los proyectos reales que han inspirado el método de trabajo presentado en esta tesis.

- No es conveniente por complicar mucho la implantación y hacer más lento el proceso de evaluación, ya que exige un algoritmo de control de convergencia preparado para cualquier combinación posible de variables independientes.

Lo que se propone aquí es una solución intermedia. Por un lado, se podrá fijar el valor de cualquier variable independiente, pues esto siempre simplifica la búsqueda sin crear problemas de convergencia en los cálculos. Por otro lado, se puede permitir un número reducido de alternativas en la selección del conjunto de variables independientes, estableciendo un control de convergencia propio para cada alternativa. Esto última solución se aplicará sólo en aquellos problemas cuya naturaleza realmente lo exija.

La segunda dificultad, o sea, la **lentitud** de los cálculos, puede ser crítica en un proceso iterativo como la optimización del diseño de sistemas complejos. En el Capítulo 7 (sustitución de funciones lentas por ábacos) se propone una línea de trabajo original de esta tesis, desarrollada para reducir drásticamente el tiempo empleado en evaluar diseños en proyectos reales.

Nota bibliográfica:
Ejemplos del uso de modelos simplificados en fases iniciales del diseño son: [Cuadra et al, 85], [García et al, 87], [Agogino & Almgren, 87] y [Dietterich & Ullman, 87].
Un ejemplo muy claro del uso de distintos niveles de simplificación para distintas fases de diseño es el diseño de circuitos VLSI, como ya se ha comentado (por ejemplo, véase [Kuh & Ohtsuki, 90]).

En el caso ejemplo de la reactancia, y para el nivel simplificado de modelado propio de la primera fase de diseño, el **proceso de evaluación** se puede describir así (ver apartado 3.2):

i) Para cada nuevo caso de diseño planteado:

Dados los requisitos de diseño $\{ L_0, a, b, \partial_{max}, I_n \}$:

- Se halla el valor de r_{min} en función de $\{ a, b \}$:

$$r_{min} := r_{min}(a,b)$$

ii) Para cada nuevo punto que se desee evaluar:

Dados los valores de las variables independientes: $X = \{ m, n, r \}$

- Se halla el valor de ∂ y el de e_∂ (ver apartado 3.3):

$$\partial = \partial(a, b, I_n \mid m)$$
$$e_\partial = e_\partial(\partial_{max} \mid \partial)$$

- Se halla el valor de L y el de e_L :

$$L = L(a, b \mid m, n, r)$$
$$e_L = e_L(L_0 \mid L)$$

- Se halla el valor de e_r :

$$e_r = e_r(r_{min} \mid r)$$

- Se halla el valor de p :

$$p = p(a, b, I_n \mid m, n, r)$$

- Se halla el valor de v :

$$v = p(a, b \mid m, n, r)$$

iii) Para cada nueva función objetivo:

Dados los factores de peso de la función objetivo $f_i(X)$:

$Wi = \{ wp_i , wv_i , wL_i , w\partial_i , wr_i \}$

- Se halla el valor de la función objetivo:

$$f_i(X) = wp_i\ p + wv_i\ v + wL_i\ e_L + w\partial_i\ e_\partial + wr_i\ e_r$$

La figura 3.8 muestra este proceso de evaluación usando el método de representación ANA. En la figura no se han representado las iteraciones de la optimización, pues se intenta sólo mostrar el proceso de evaluación y además con respecto a una sola función objetivo. En realidad, para cada nuevo caso de diseño planteado se podrán evaluar muchos puntos distintos, y cada uno de ellos con respecto a varias funciones objetivo.

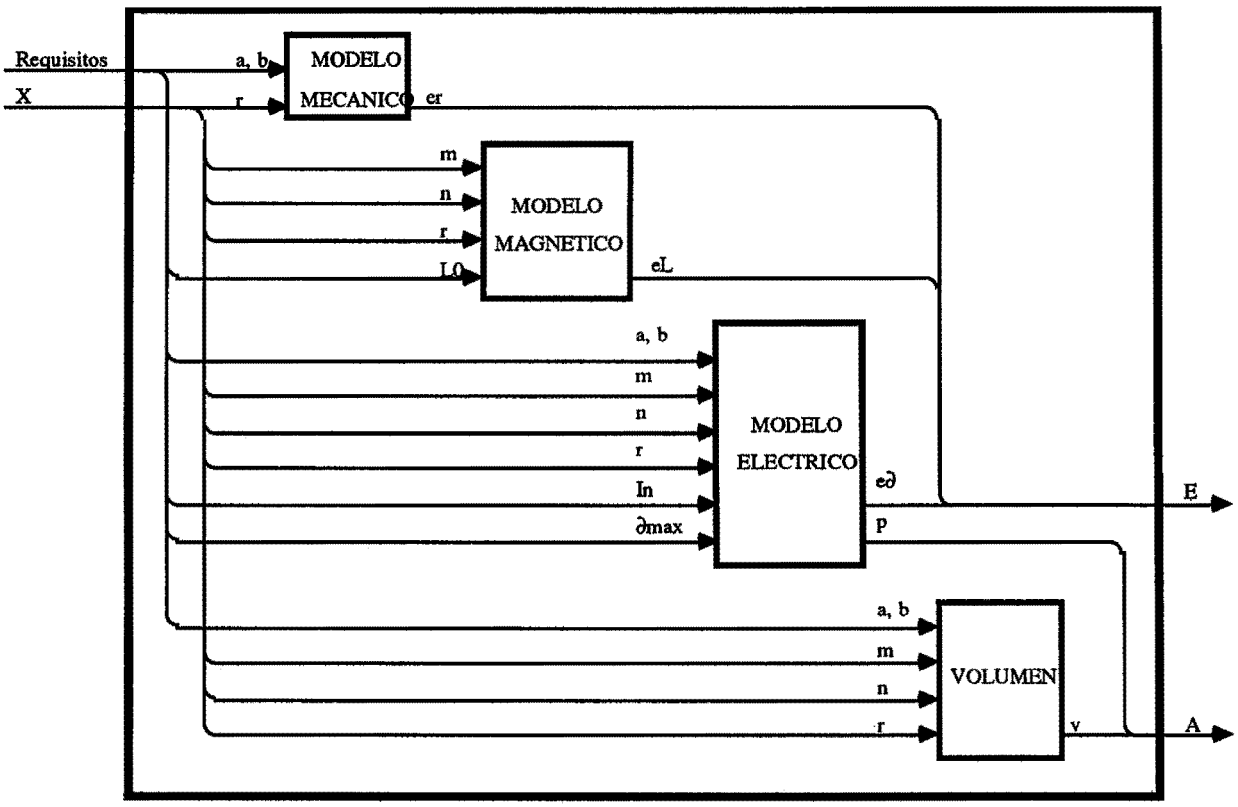


Figura 3.8. Proceso de Evaluación de la Fase 1 del Diseño de la Reactancia

El **proceso de búsqueda** (ver figura 3.7) es el encargado de cambiar el valor de las variables independientes (X) sin salirse de la región de búsqueda (**región**) en función de:

- Los puntos anteriores (X).
- Las características de los puntos previos (XX).
- La definición del problema (**requisitos**).

El número de variables independientes puede ser muy elevado, y además puede variar según el valor de alguna variable (como en el caso ejemplo del apartado 3.2). Además, la naturaleza matemática de las variables puede ser muy diversa, lo que hace que el espacio de búsqueda sea poco regular. Estas complicaciones, entre otras, hacen que la resolución eficaz de un problema complejo necesite de una optimización estructurada, descomponiendo el proceso de búsqueda en un sistema de búsquedas u optimizaciones parciales.

En el **Capítulo 4** se describe con detalle cómo puede llevarse a cabo **una fase de diseño**, es decir, una búsqueda sistemática capaz de hallar óptimos locales y globales en regiones de búsqueda complejas, con distintos tipos de variables y en espacios de dimensión variable, por medio de una estructura de **optimizaciones parciales**.

El **Capítulo 5** está dedicado a las **estrategias de búsqueda** que se acomodan a los distintos tipos de variables independientes: continuas, discretas y muy discretas, es decir, está dedicado a cómo resolver cada una de las optimizaciones parciales. En él se describen especialmente las aportaciones de la tesis, tanto a nivel teórico (nuevos desarrollos) como a nivel de aplicación de algoritmos ya conocidos a proyectos reales.

En el **Capítulo 6** se trata la forma de abordar las optimizaciones **multiatributo**. En particular, las búsquedas locales multiatributo se pueden sustituir por varias búsquedas monoatributo. Recuérdese que las búsquedas locales se suelen emplear cuando se han de resolver problemas con variables continuas.

El **Capítulo 7** está dedicado al uso y creación automática de **ábacos** (funciones rápidas) capaces de sustituir a procesos de cálculo lentos. Es un tema totalmente original de esta tesis que ha surgido ante la necesidad real de incorporar procesos de cálculo lentos a programas con una estructura fuertemente iterativa, como los que resultan de aplicar el modelo de optimización de diseño expuesto en este trabajo. Se presentan en ese capítulo métodos de interpolación en espacios de n dimensiones que se desarrollaron expresamente para ser aplicados en proyectos reales.

En el **Capítulo 8** se presentan las conclusiones y aportaciones de la tesis y se citan líneas de trabajo para futuros desarrollos; en especial se apuntan técnicas aplicables a la fase de

análisis de consistencia y prediseño, muy apoyada en el conocimiento heurístico y muy dependiente del problema específico de aplicación.

3.5 - Notas sobre Aplicaciones a Proyectos.

Las ideas y metodologías expuestas en este capítulo están inspiradas en el trabajo llevado a cabo en proyectos reales de investigación y desarrollo. En este apartado se comenta el trabajo realizado en dichos proyectos a la luz de las ideas y la nomenclatura utilizadas aquí, como si éstos fueran consecuencias de la tesis y no al contrario. De esta forma las aplicaciones reales proporcionan un respaldo experimental a las ideas y metodologías aquí expuestas. Con este objeto el orden de las notas no es cronológico, sino que se adapta al orden de exposición del capítulo.

La notación seleccionada para el problema general de diseño tiene su origen en los problemas de optimización matemática planteados en [Pérez Arriaga, 78], [Cuadra et al, 85] y [García et al, 87]; en estos proyectos aparecen ya los conceptos de variables independientes, ligaduras, restricciones, penalización de restricciones con factores de peso, función objetivo, óptimo local y óptimo global, búsqueda y evaluación.

En [IIT, 87] se introdujeron los conceptos de atributo, espacio de atributos y optimización multiatributo. En [Cuadra & León, 90] y [Cuadra et al, 90] se completó la formalización del problema, añadiéndose los conceptos de región de búsqueda y espacio de restricciones y las definiciones de "óptimo local" y "requisitos" que aquí se proponen.

En [Cuadra et al, 85] ya se llevó a cabo una descomposición del proceso de diseño en fases y de cada fase en una estructura de optimizaciones parciales anidadas y paralelas. Sin embargo, hasta [Cuadra & León, 90] y [Cuadra et al, 90] no se formalizó y definió este proceso de descomposición estructurada con precisión. Los términos "análisis de consistencia y prediseño" e "interfaz final" se utilizaron por primera vez en [IIT, 87]. La definición de la metodología global como "Optimización Estructurada Multiatributo" se presenta por primera vez en esta tesis.

En [Cuadra et al, 85] se encuentran ya los elementos fundamentales de una optimización estructurada, aunque en este caso fuera monoatributo. El proyecto consistía en desarrollar una

herramienta informática que optimizara (con respecto al coste de fabricación) el diseño de "reactancias limitadoras de corrientes de falta en líneas eléctricas", respetando los requisitos impuestos por el cliente. En el apartado 3.2 se describe una versión muy simplificada del problema que se utiliza luego como ejemplo a lo largo del Capítulo 3.

El problema reunía todas las dificultades citadas en el apartado 3.4 excepto el ser multiatributo. El número de variables era elevado, oscilando entre 10 y 30 según los casos. La dimensión del vector de variables dependía del valor de las variables más importantes (ver apartado 4.3). Había variables independientes muy discretas, discretas y continuas mezcladas. el número de restricciones era elevado (entre 10 y 13), sin contar las correspondientes a los rangos de las variables independientes. Los procesos de cálculo eran lentos, en especial el modelo electromagnético y el modelo térmico.

Se estableció una jerarquía de tres niveles para las variables independientes. Los criterios empleados en la jerarquización fueron tanto la necesidad de llevar a cabo todas las búsquedas en espacios de dimensión constante (ver apartado 4.3) como el "tamaño" de las variables independientes (muy discretas, discretas y continuas; ver apartado 5.1).

El algoritmo constaba de tres fases de diseño correspondientes a otros tantos niveles de detalle en el modelado; al final de la primera fase la variable vectorial de nivel más alto (muy discreta) quedaba fijada en el valor óptimo obtenido, a la vez que se ofrecía un buen punto inicial para la segunda fase; al final de la segunda fase la variable vectorial de segundo nivel (discreta) quedaba también fijada en el óptimo obtenido, ofreciendo a su vez un buen punto inicial para la tercera fase.

La primera fase de diseño estaba compuesta por dos optimizaciones parciales anidadas, la de nivel superior con variables muy discretas y la de nivel inferior con variables discretas y continuas; la segunda fase estaba compuesta también por dos optimizaciones parciales anidadas, la superior con variables muy discretas y la inferior con variables continuas; la tercera fase de diseño estaba compuesta por dos optimizaciones parciales paralelas, ambas con variables continuas.

En [García et al, 87] se resolvió un problema similar, aunque con ciertas características especiales que sugirieron cambios en la selección y jerarquía de las variables independientes. Los algoritmos de búsqueda se mejoraron y se desarrollaron módulos de cálculo más precisos, especialmente en los modelos térmico y electromagnético (ver figuras 3.8, 4.7).

Ambos proyectos se programaron en FORTRAN 77, obteniéndose resultados excelentes en tiempos de cálculo razonables (entre 1 y 10 minutos de CPU en un ordenador Data General ECLIPSE MV/8000). No se detectaron problemas de mínimos locales, y el tiempo consumido en la obtención de soluciones variaba coherentemente con la complejidad de las soluciones obtenidas (más bobinas en paralelo implicaban más variables independientes) en función de los requisitos del usuario.

En [IIT, 87] se realizó un estudio de viabilidad de la aplicación de CAD al diseño de vehículos espaciales. El problema fundamental residió en la adquisición de conocimiento sobre satélites artificiales e industria espacial en general y la revisión de técnicas para diseñar grandes sistemas. Dado que era un estudio puramente teórico sólo se implantó un prototipo muy simple, que no obstante presentaba ciertas dificultades: variables muy discretas mezcladas con continuas, dos atributos conflictivos con respecto a los cuales optimizar y módulos de cálculo extremadamente lentos.

El prototipo resolvía el problema de optimizar los parámetros de la órbita (circular) de un satélite artificial y algunas variables relativas a la configuración del mismo, con respecto a dos atributos: coste total (fabricación y puesta en órbita) y fiabilidad. El prototipo cumplió su tarea, pero la existencia de un módulo de cálculo muy lento (de 3 a 4 segundos por evaluación, necesitándose varios cientos de evaluaciones) hizo que los resultados no fueran espectaculares de cara al usuario. De este problema concreto surgió la idea de los ábacos de interpolación lineal (ver Capítulo 7), y gracias al proyecto en su conjunto se establecieron las bases de la metodología expuesta en esta tesis.

El problema planteado en [Cuadra & León, 90] era de naturaleza muy distinta. Se trataba de distribuir óptimamente la carga del transbordador espacial Hermes. Se implantó, a modo de prototipo, la solución del subproblema principal: la distribución de un conjunto de objetos en el interior de un cajón de dimensiones dadas, siendo los objetos cajas y cilindros de dimensiones cualesquiera. El atributo de optimización era maximizar el aprovechamiento del cajón en cuanto a volumen ocupado (volume efficiency). Cada objeto llevaba asociado un conjunto de restricciones que limitaban sus posibles orientaciones y colocación en el cajón.

El problema de distribución de objetos reunía las siguientes dificultades: un número de variables independientes elevado (qué objetos se colocan, en qué posición cada uno y en qué orientación); el número de variables independiente era variable; variables independientes discretas (qué objeto, en cuál de las seis posibles orientaciones) mezcladas con variables

continuas (en qué posición); un número elevado de restricciones (hasta tres por objeto); además de todo esto era difícil evaluar la colocación secuencial de cada objeto hasta tener el cajón lleno.

El problema se resolvió mediante una búsqueda en árbol organizada recursivamente (búsqueda en profundidad), donde en cada nivel de recursión las ramificaciones del árbol representaban alternativas jerarquizadas: 1) qué objeto se coloca a continuación, 2) (para cada objeto) en qué orientación, 3) (para cada objeto ya orientado) en qué posición.

Para evitar la explosión combinatoria, en cada nivel del árbol se realizaba una selección multiatributo (ver Capítulo 6, y en especial la figura 6.9). Uno de los atributos era "real", el volumen total colocado hasta el momento, y el otro un heurístico que estimaba el volumen que quedaba aprovechable.

Por diversas razones el proceso se dividió en dos fases de optimización: la primera, encargada de llenar lo mejor posible el fondo del cajón; la segunda, encargada de llenar lo mejor posible el resto del cajón, partiendo de todas y cada una de las soluciones ofrecidas por la primera fase. Las soluciones finales de la primera fase eran seleccionadas con criterio multiatributo.

Aparte de las dos fases de optimización se implantó un "interfaz inteligente" muy sencillo encargado de las entradas de datos (teclado y/o fichero) y cierto preprocesado de estos, como ordenar los objetos según criterios heurísticos y formalizar adecuadamente las restricciones asociadas a dichos objetos. También se implantó un "interfaz final" encargado de informar al usuario de las soluciones y de la marcha del proceso, así como de generar informes escritos. El interfaz final también realizaba una tarea fundamental, pues generaba ficheros compatibles con herramientas gráficas de CAD ya existentes (AUTOCAD y ESABASE), de tal manera que el usuario pudiera ver e imprimir la distribución espacial resultante.

La implantación fue un éxito. El algoritmo se programó en lenguaje C. Corrió en un ordenador IBM PS/50, obteniendo soluciones excelentes en tiempos de 30 a 120 segundos de CPU. El programa controlaba dinámicamente las reservas de memoria de la máquina y se adaptaba al tiempo máximo de respuesta, que era un requisito impuesto por el usuario.

El problema planteado en [Cuadra et al, 90] reunía todas las dificultades posibles relacionadas con variables, restricciones, atributos y módulos de cálculo. Consistía en optimizar el diseño de un motor de reluctancia variable (VRM, Variable Reluctance Motor)

incluyendo el equipo electrónico (inversor) encargado de alimentarlo y el equipo encargado de su control de velocidad.

El proyecto se planteó simplemente como ejemplo de aplicación de la metodología expuesta aquí, preparando una aplicación real posterior. Por ello no se llegó a la fase de implantación informática, pero se realizó una gran labor de adquisición del conocimiento y modelado del problema, teniendo en cuenta que de nuevo era un problema absolutamente desconocido. Se plantearon dos fases posibles de diseño, aparte de una fase muy interesante de análisis de consistencia y prediseño basada en modelos muy sencillos del motor.

Como productos de este trabajo se pueden citar la formalización completa del problema general de diseño, el concepto de estructura de optimizaciones paralelas y anidadas y una aplicación muy interesante de los ábacos de interpolación lineal, pues se implantaron en un sólo ábaco varias tablas de doble entrada que describían los parámetros del circuito magnético formado por los dientes del rótor y el estátor (tanto alineados como desalineados).

Capítulo 4

Optimizaciones Parciales

La optimización de diseño en Ingeniería es una tarea muy compleja que requiere de una visión precisa y global del sistema concreto a diseñar y de conocimientos generales sobre procedimientos de optimización. En los capítulos anteriores se proporcionó una descripción formal del problema y de sus elementos, destacando algunos de éstos para ser abordados con más detalle.

En este capítulo se trata el proceso central de dicho problema: cómo se pueden explorar grandes espacios de variables independientes con objeto de buscar todos los óptimos locales o las soluciones de interés, según se emplee una búsqueda local o una búsqueda en árbol. La solución que se formula y desarrolla aquí como parte fundamental de la tesis es la descomposición de una optimización en **optimizaciones parciales.**

La descomposición de un problema complejo de optimización en subproblemas de optimización más sencillos no es, evidentemente, una idea original de esta tesis. Lo que sí es original, al menos con respecto a la bibliografía revisada, es la organización sistemática de un problema según una estructura de **optimizaciones anidadas y optimizaciones paralelas**, y especialmente los criterios sugeridos para definir dicha estructura; dichos criterios se basan fundamentalmente en el conocimiento específico sobre las variables independientes y sus características.

En el apartado 4.1 se describen los subproblemas principales que forman una fase de diseño; en el apartado 4.2 se trata la tarea básica de una fase de diseño, que es la optimización monoatributo; en el 4.3 se propone la descomposición de una optimización monoatributo compleja en un sistema estructurado de optimizaciones parciales. Finalmente, el apartado 4.4

está dedicado a comentar aspectos relacionados con este capítulo de los proyectos reales de diseño por ordenador que han dado lugar a esta tesis.

En el Capítulo 5 se tratará la forma de resolver cada optimización parcial en función de sus características propias, y especialmente en función de las propiedades del subconjunto de variables independientes de las que se ocupa la optimización parcial.

4.1 - Solución de una Fase de Diseño.

La misión de una fase de diseño es encontrar un conjunto de puntos del espacio de variables independientes tales que sean una muestra de los puntos pertenecientes a la hipersuperficie óptima del espacio de atributos. Para ello se parte de un conjunto de puntos iniciales pertenecientes a distintas regiones de búsqueda, proporcionados por los resultados de la fase de diseño anterior.

En el apartado 3.4 del capítulo anterior se describió la estructura interna de una fase de diseño genérica, mostrando su descomposición en subproblemas y los intercambios de datos entre los mismos (ver figura 3.7).

Los subproblemas de una fase que se han mencionado son el interfaz, el proceso de evaluación, el proceso de búsqueda y la selección multiatributo:

- El **interfaz** se limita a realizar un cambio de variables para trasladar puntos de un espacio de variables independientes a otro.

- El **proceso de evaluación** es el encargado de calcular todas las variables de diseño de interés a partir del valor de las variables independientes. Los problemas de convergencia y de lentitud en los cálculos que pueden aparecer en este proceso serán tratados en otros capítulos de la tesis.

- El **proceso de búsqueda** es el encargado de encontrar puntos cada vez más "interesantes" según las restricciones y los atributos. Para ello debe cambiar el valor de las variables independientes sin salirse de la región de búsqueda, teniendo en cuenta los

puntos anteriores, las características de los puntos anteriores y la definición del problema (requisitos).

- La **selección multiatributo** es el proceso que decide qué puntos "interesantes" encontrados por el proceso de búsqueda se escogen como muestra de la hipersuperficie óptima del espacio de atributos. Los criterios de selección se exponen en el Capítulo 6.

Se ha mencionado ya que en esta tesis se consideran dos grandes familias de estrategias de búsqueda: la búsqueda local y la búsqueda en árbol. En una búsqueda local se sigue un sólo criterio de optimización cada vez (una sola función objetivo, aunque considere varios atributos), mientras que en una búsqueda en árbol se pueden seguir varios criterios de optimización simultáneamente por distintas ramas del árbol (ver Capítulo 5).

Para realizar una optimización multiatributo por medio de búsquedas locales (ver Capítulo 6) se puede aproximar la búsqueda a la hipersuperficie óptima por medio de planos tangentes a la misma. Cada vector normal a un plano señala una dirección de búsqueda y por tanto representa un criterio de optimización. Cada búsqueda llegará a un óptimo local en general distinto, y la selección multiatributo eliminará los óptimos locales que no parezcan pertenecer a la hipersuperficie óptima.

Por medio de búsquedas en árbol se puede avanzar simultáneamente en varias direcciones de optimización. Debido a limitaciones de memoria, se eliminarán dinámicamente los puntos (nodos del árbol) que parezcan más alejados de la superficie óptima, expandiendo preferentemente los nodos que parezcan más cercanos. Como se puede ver la selección multiatributo se aplica durante todo el proceso de búsqueda y no sólo al final del mismo.

En cualquier caso, se puede afirmar que la tarea básica en una fase de diseño consiste en avanzar por el espacio de variables independientes en una dirección tal que se produzcan avances deseados en el espacio de atributos y en el espacio de estado de las restricciones. Esta tarea básica se presenta tanto en las búsquedas locales como en las búsquedas en árbol, y se denomina aquí **optimización monoatributo**.

4.2 - Optimización Monoatributo.

Como se ha visto en el apartado anterior, una fase de diseño (en un caso genérico) consiste en un conjunto de optimizaciones monoatributo. Para llevar a cabo una **optimización monoatributo** es necesario definir:

- Una región de búsqueda.
- Un punto inicial.
- Una función objetivo (que incluye atributos y restricciones).
- Una estrategia de búsqueda.

La optimización monoatributo se considera finalizada cuando se alcanza un óptimo local según la función objetivo y la estrategia correspondientes. La figura 4.1 muestra los dos procesos fundamentales de una optimización monoatributo (evaluación y búsqueda) y los intercambios de datos correspondientes. En lo sucesivo, a no ser que se especifique lo contrario, se utilizará el término **optimización** para indicar **optimización monoatributo.**

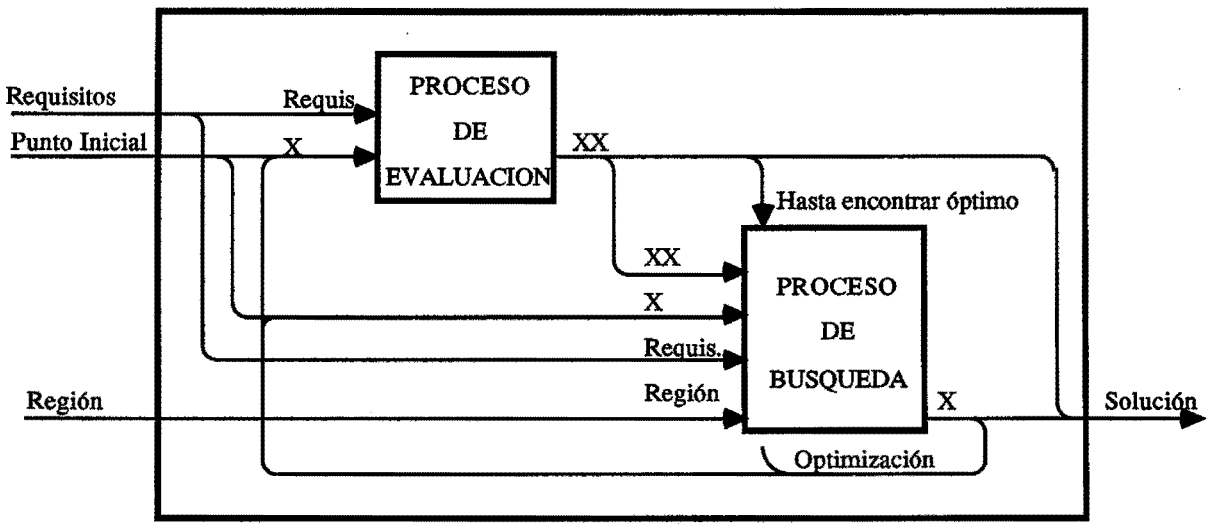


Figura 4.1. Optimización Monoatributo.

Si un sistema es complejo, el número de grados de libertad en su diseño puede ser alto; esto implica un número elevado de variables independientes y por tanto un espacio de búsqueda grande. Además del número elevado de dimensiones, la búsqueda en este espacio puede presentar **complicaciones adicionales** (véase por ejemplo [Galiana & McGillis, 88]):

- Las variables independientes **no son homogéneas** en cuanto a su naturaleza matemática; aparecen mezcladas en el espacio de búsqueda variables continuas, discretas y, como ya se verá más adelante, variables muy discretas.

- Todas las variables independientes **no son igualmente importantes** en cuanto a su influencia en el diseño; suele haber una gama muy variada y difícil de matizar de importancias relativas de unas frente a otras.

- La **dimensión** del espacio de búsqueda puede ser **variable**, si dicha dimensión depende del valor concreto que tome alguna otra variable.

- La **función objetivo** tendrá en general una forma extraña, estando **redefinida** y llena de **mínimos locales** (recuérdese que en [Rodríguez-Piñero, 86] se llama función redefinida a la que tiene distintas leyes de obtención de imágenes en distintas regiones de su dominio). Esto es debido fundamentalmente a la existencia de variables discretas y a la influencia del conjunto de restricciones en la definición de la función objetivo.

- El **conocimiento** que el ingeniero utiliza para guiar eficazmente la búsqueda es muy **difícil de recoger** exhaustivamente, por estar lleno de matizaciones y casos particulares. El tipo de conocimiento manejado es tal que permite anticipar cualitativa y cuantitativamente el impacto que ciertos cambios en las variables independientes producirán en un diseño provisional concreto, mejorando así sus características.

La mayoría de estas complicaciones se pueden resolver de forma conjunta con la descomposición en **optimizaciones parciales** que se expondrá en este mismo capítulo. Este método se ha utilizado ya en varios proyectos reales, como [Cuadra et al, 85] y [García et al, 87], y tiene su origen en la necesidad de resolver optimizaciones en espacios de búsqueda de dimensión variable.

El objetivo es descomponer una optimización muy compleja en un conjunto estructurado de optimizaciones sencillas, llamadas **optimizaciones parciales**, cada una de las cuales no presenta las dificultades mencionadas. Dentro de la aplicación específica de optimización de diseño, las optimizaciones parciales serán también llamadas **bucles de diseño**.

4.3 - Descomposición en Optimizaciones Parciales.

Es muy frecuente que, al intentar resolver una optimización, el espacio de variables independientes presente algunas de las dificultades ya mencionadas, es decir, la de ser de dimensión variable, la falta de homogeneidad en la naturaleza matemática de las variables independientes y la falta de homogeneidad también en la importancia relativa de dichas variables.

Una optimización en un espacio de búsqueda de dimensión variable se puede resolver por medio de la **descomposición en optimizaciones parciales**. Sea X un vector de variables independientes de dimensión variable; si se puede descomponer X en dos subespacios $X = \{ X_1, X_2 \}$, tales que:

- La dimensión de X_1 es constante.
- La dimensión de X_2 depende del valor que tome X_1.

Entonces el problema se puede resolver como dos optimizaciones parciales en espacios de dimensión constante, donde la optimización de X_2 se realiza para cada valor propuesto de X_1. La figura 4.2 muestra el esquema del proceso conjunto.

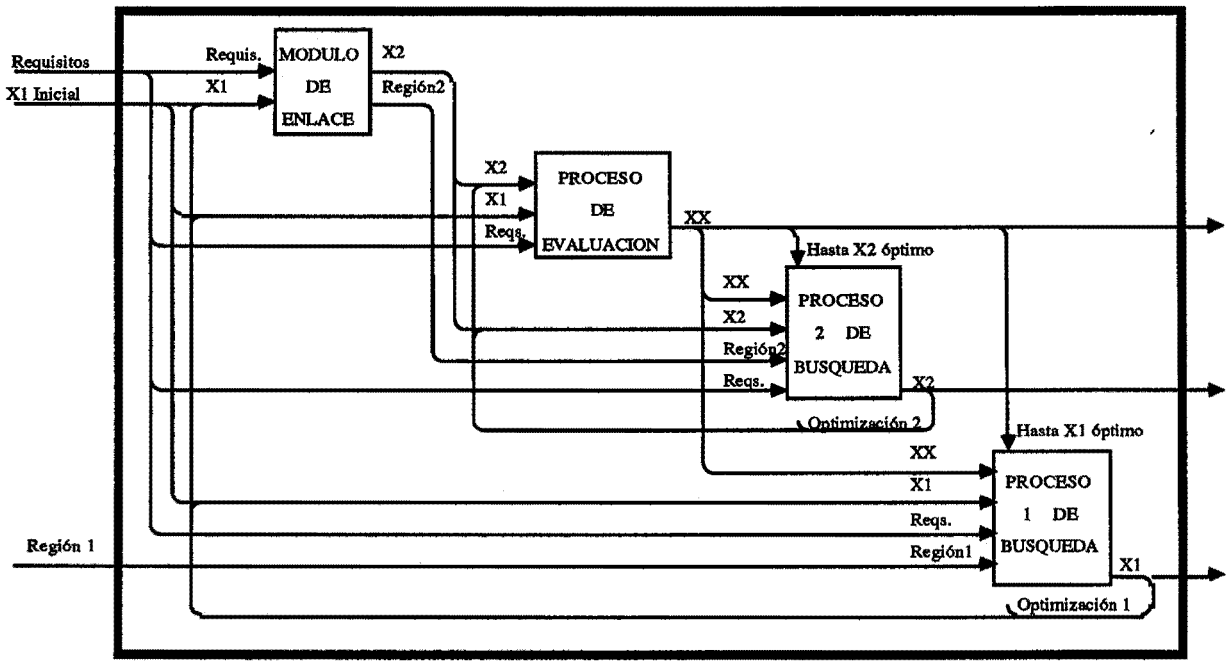


Figura 4.2. Descomposición de una Optimización en Dos Optimizaciones Parciales

Como se puede ver en la figura 4.2, hay dos optimizaciones anidadas y un módulo de enlace entre ambas. El bucle 2 optimiza el valor de X_2 para cada valor de X_1 según los resultados obtenidos en el proceso de evaluación; el bucle 1 optimiza el valor de X_1 según los resultados obtenidos en el proceso de optimización del bucle 2. El **módulo de enlace** se encarga de obtener buenos puntos iniciales para X_2 en función del valor de X_1; para ello puede utilizar modelos menos detallados del sistema a diseñar, tomados (por ejemplo) de fases de diseño anteriores.

Es ahora necesario matizar la diferencia existente entre variable y parámetro. Ya se vio que en [Sacristán, 64] una variable se considera un símbolo que puede ser sustituido por diversos valores numéricos de una determinada clase. Según la misma fuente, un **parámetro** es:

> "... un símbolo que tiene oficio de constante para cada subclase de la clase de objetos a los que es aplicable, pero es variable en el sentido de que no tiene un constituyente único para todas las subclases de dicha clase."

En otras palabras, un parámetro es una variable que se mantiene constante temporalmente en un determinado contexto, pero que es variable con respecto a un contexto más amplio. En el ejemplo anterior, X_1 es variable en el bucle 1, pero es un parámetro en el contexto del bucle 2. La dimensión del espacio X_2 es también constante durante el bucle 2, pues depende únicamente del valor de X_1.

En el ejemplo del Capítulo 3 el espacio de variables independientes es de dimensión variable:

$$X = \{ m, n_1, n_2, ..., n_m, r \}$$

Para resolverla se podría descomponer el problema en dos optimizaciones parciales. En la primera se optimiza m, y en la segunda se optimiza el resto de las variables para cada valor propuesto de m (una optimización para cada valor distinto de m). De esta forma m es constante (un parámetro) en las segundas optimizaciones, y por tanto cada una de éstas se realiza en un espacio de dimensión constante. Las variables independientes quedarían:

- Primera optimización: $X_1 = \{ m \}$
- Segunda optimización: $X_2 = \{ n_1, n_2, ..., n_m, r \}$

El caso de dimensión variable es un ejemplo claro donde se ve la necesidad de descomponer una optimización en optimizaciones parciales. En otros casos puede también resultar muy ventajosa una descomposición similar:

- Cuando X_1 y X_2 son de naturaleza matemática distinta (por ejemplo, X_1 discreta y X_2 continua).

- Cuando la importancia de X_1 es mucho mayor que la de X_2 respecto a su impacto en el diseño.

- Cuando los valores factibles de X_2, y por tanto sus valores iniciales, dependen fuertemente del valor de X_1.

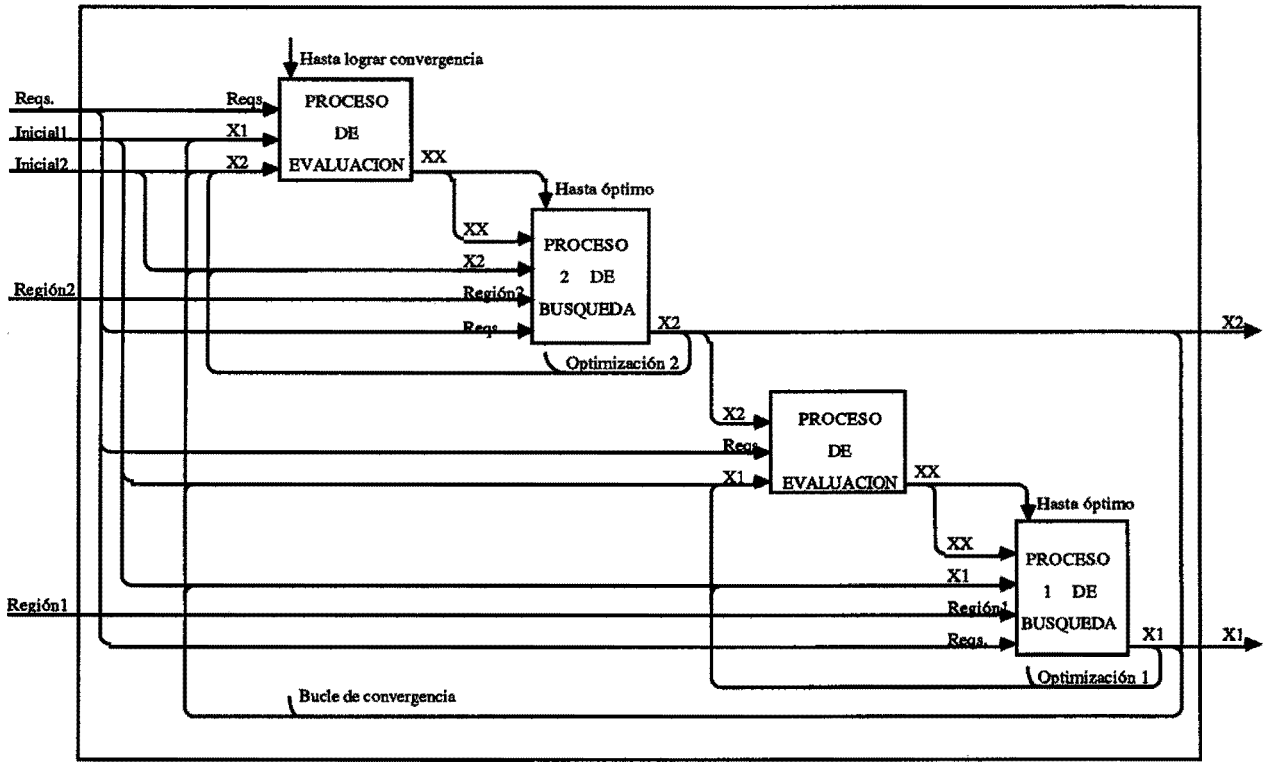


Figura 4.3. Dos Optimizaciones Paralelas

Todo apunta hacia la posibilidad de una **jerarquización de las variables independientes**, atendiendo a criterios como la dimensión constante de cada subespacio de variables, la naturaleza matemática de dichas variables y su importancia relativa. Cuando las

variables independientes admiten una jerarquía así, la optimización se puede descomponer en un conjunto de **optimizaciones parciales anidadas.**

También puede suceder que un vector de variables independientes X admita ser descompuesto en dos vectores de jerarquía semejante, X_1 y X_2, tales que el valor óptimo de (al menos) uno de ellos dependa poco del valor que tenga el otro. En este caso se podrá descomponer la optimización en dos **optimizaciones parciales paralelas** garantizando la convergencia, esto es: se optimiza X_1 manteniendo constante X_2 (como parámetro) y luego se optimiza X_2 manteniendo X_1 constante, y así sucesivamente hasta conseguir que converja a una solución óptima conjunta.

La figura 4.3 representa la estructura de dos optimizaciones parciales paralelas. Puede observarse la diferencia entre esta estructura y la de las optimizaciones anidadas de la figura 4.2, aunque esta diferencia quedará más clara en los apartados siguientes. Un rasgo común de los dos tipos de descomposición es que cada optimización parcial contará con sus propias estrategias de búsqueda.

4.3.1 - Optimizaciones Parciales Anidadas.

Sea un espacio de variables independientes X en el que se ha establecido una jerarquía de variables:

$$X = \{ X_0, X_1, X_2 \}$$

tal que:

X_0 es de nivel superior a X_1

X_1 es de nivel superior a X_2

Sea f(X) la función objetivo con respecto a la cual se desea optimizar el valor de X. La **optimización parcial anidada** de X_1 según f(X) consiste en:

Hallar el óptimo de $f(X_0, X_1, X_2)$

tal que:

X_0 es constante (es un parámetro) durante toda la optimización parcial.

X_2 es el resultado de una optimización parcial según f(X).

Esto último quiere decir que para cada valor propuesto de $\{ X_0, X_1 \}$ se halla el valor de X_2 que hace óptima f(X), tomando los valores de las variables de nivel superior (en este caso

X_0 y X_1) como parámetros. La notación empleada aquí es la siguiente: f(x | μ) es una función "f" que depende de una variable "x" y de un parámetro "μ".

El valor de X_0 se podrá optimizar parcialmente con un algoritmo iterativo que sólo se ocupe de variar los valores de X_0 según los valores de la **función objetivo parcial** $f_0(X_0)$, definida como:

$$f_0(X_0) = f(X_0, X_1, X_2) \;/\; X_1 \text{ es óptimo parcial según } f_1(X_1) \;/\; X_2 \text{ es óptimo parcial según } f_2(X_2)$$

El algoritmo responsable de la optimización parcial de X_1 sólo tiene que ocuparse de variar iterativamente los valores de X_1 según los valores de una función objetivo parcial $f_1(X_1)$, perteneciente a la **familia de funciones** (pues depende de parámetros) definida como:

$$f_1(X_1 \mid X_0) = f(X_0, X_1, X_2) \;/\; X_0 \text{ es constante (parámetro)} \;/\; X_2 \text{ es óptimo parcial según } f_2(X_2)$$

siendo $f_2(X_2)$ una función de la **familia de funciones** objetivo parciales (una función para cada valor de { X_0, X_1 }) definida como:

$$f_2(X_2 \mid X_0, X_1) = f(X_0, X_1, X_2) \;/\; X_0 \text{ es constante (parámetro)} \;/\; X_1 \text{ es constante (parámetro)}$$

Se puede comprobar fácilmente que al obtener así el óptimo X_0 se obtiene a la vez el óptimo X, pues se tiene el mejor valor de X_2 para el mejor valor de X_1 para el mejor valor de X_0. La figura 4.4 representa un proceso de optimizaciones parciales anidadas con sólo dos niveles de vectores de variables, X_1 y X_2.

Para que la descomposición en optimizaciones parciales anidadas sea eficaz es necesario contar con procesos auxiliares que permitan hallar buenos puntos iniciales para cada optimización parcial, en función naturalmente del valor de las variables de nivel superior (parámetros en el contexto particular de dicha optimización parcial).

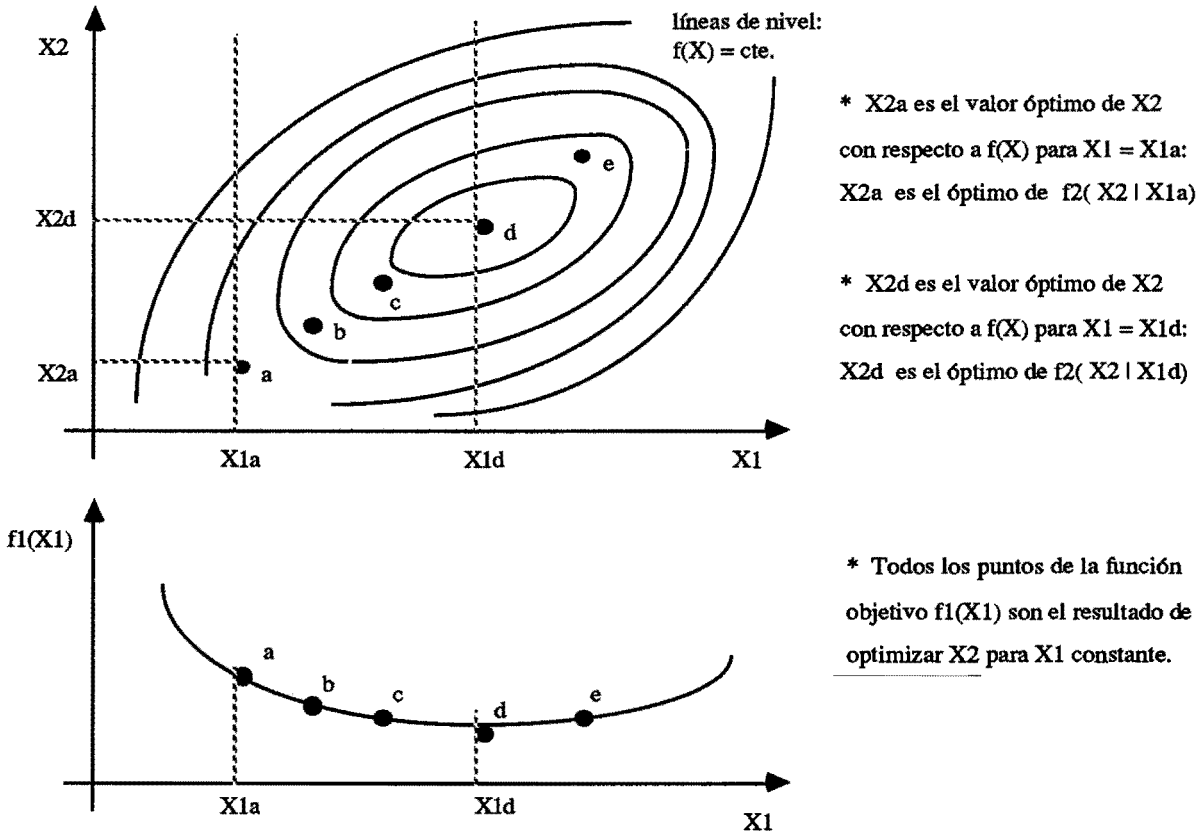


Figura 4.4. Dos Optimizaciones Parciales Anidadas

4.3.2 - Optimizaciones Parciales Paralelas.

Sea un espacio de variables independientes X en el que se han distinguido dos subespacios de variables:

$$X = \{ X_1, X_2 \}$$

tales que:

el valor óptimo de X_1 depende poco del valor que tome X_2

el valor óptimo de X_2 puede o no depender del valor que tome X_1

Sea f(X) la función objetivo con respecto a la cual se desea optimizar el valor de X. Dados unos valores iniciales para X_1 y X_2, las **optimizaciones parciales paralelas** de X_1 y X_2 según f(X) consisten en:

Hallar el óptimo de $f(X_1, X_2)$

siguiendo los pasos:

i) Se optimiza X_2 manteniendo X_1 constante.

ii) Se optimiza X_1 manteniendo X_2 constante (el valor obtenido en (i)).

iii) Se vuelve al paso (i) hasta que X_1 y X_2 converjan a una solución óptima.

El algoritmo responsable de la optimización parcial de X_1 sólo tiene que ocuparse de variar iterativamente los valores de X_1 según los valores de una **función objetivo parcial** $f_1(X_1)$, perteneciente a la **familia de funciones** definida como:

$$f_1(X_1 \mid X_2) \quad = \quad f(X_1, X_2) \quad / \quad X_2 \text{ es constante (parámetro)}$$

Del mismo modo, el algoritmo responsable de la optimización parcial de X_2 sólo tiene que ocuparse de variar iterativamente los valores de X_2 según los valores de una función objetivo parcial $f_2(X_2)$, perteneciente a la **familia de funciones** definida como:

$$f_2(X_2 \mid X_1) \quad = \quad f(X_1, X_2) \quad / \quad X_1 \text{ es constante (parámetro)}$$

La figura 4.5.a trata de representar un proceso con dos optimizaciones parciales paralelas en una función f(X) en la cual el óptimo de X_2 es prácticamente independiente del valor de X_1 y esto permite la convergencia conjunta al óptimo real.

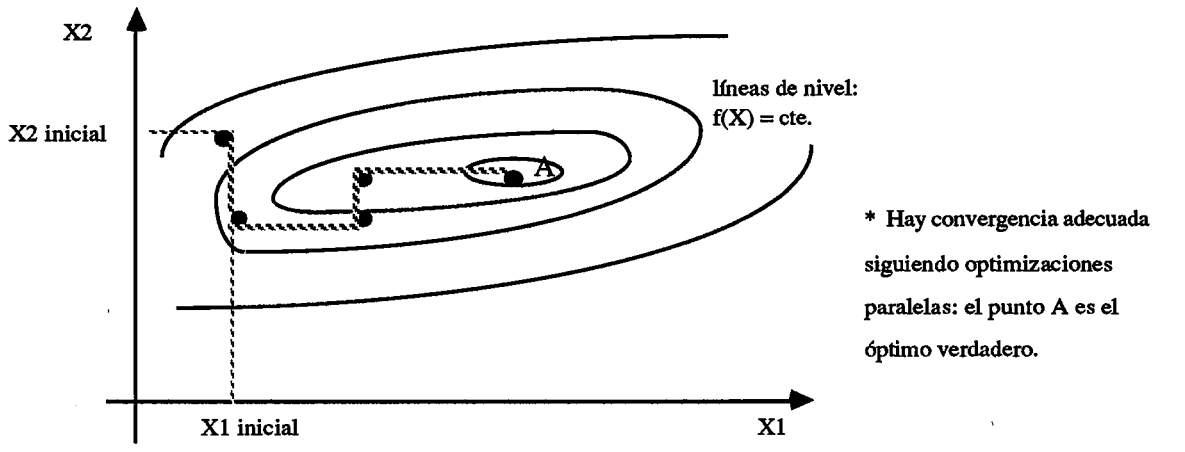


Figura 4.5.a Dos Optimizaciones Paralelas con Convergencia.

La figura 4.5.b muestra el mismo proceso pero con una función similar a la de la figura 4.4, en la cual ambas variables están más fuertemente acopladas y puede no obtenerse una

convergencia adecuada. En la práctica la convergencia depende de muchos factores, como son la forma de la región factible (restricciones), el comportamiento de la función objetivo e incluso el orden en que se realicen las optimizaciones parciales.

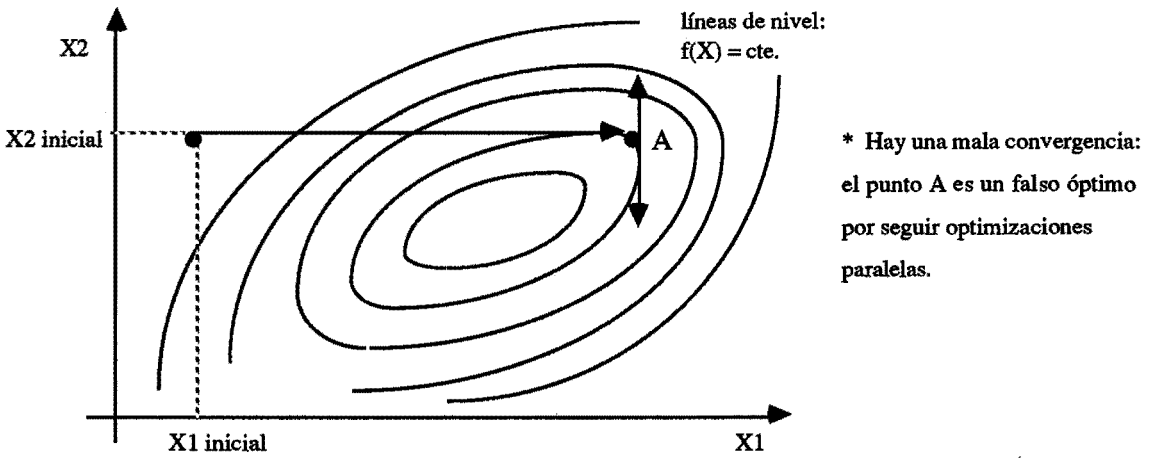


Figura 4.5.b Dos Optimizaciones Paralelas sin Convergencia.

El conocimiento específico sobre el problema de diseño es imprescindible si se quiere garantizar la convergencia adecuada de las optimizaciones paralelas. Por ejemplo, en [Cuadra et al, 90] se trata de optimizar un sistema formado por un motor eléctrico y el circuito electrónico encargado de alimentarlo por medio de dos optimizaciones paralelas. Primero se optimiza el motor suponiendo un alimentador ideal (todo componente electrónico requerido existe en el mercado); luego se optimiza el circuito electrónico con componentes reales, dejando constante el motor; después se retoca el motor dejando fijo el alimentador óptimo encontrado, y así sucesivamente hasta obtener un punto fijo en ambas optimizaciones.

4.3.3 - Estructura de un Sistema de Optimizaciones Parciales.

La figura 4.6 muestra la estructura general que presenta un proceso de optimización cuando se descompone en optimizaciones parciales. En un caso particular la estructura resultante será una combinación de módulos tales que cada uno responda a dicha estructura general.

Como se ve en la figura, el proceso de optimización global consta de un primer proceso de evaluación y una primera optimización parcial o proceso de búsqueda:

- El **primer proceso de búsqueda** (optimización parcial, bucle de diseño) se encarga de optimizar el valor de las variables independientes de más alta jerarquía.

- El **primer proceso de evaluación** se encarga de asignar uno o varios puntos de los espacios de atributos y restricciones del diseño a cada punto de la región de búsqueda del subespacio de variables independientes de más alta jerarquía.

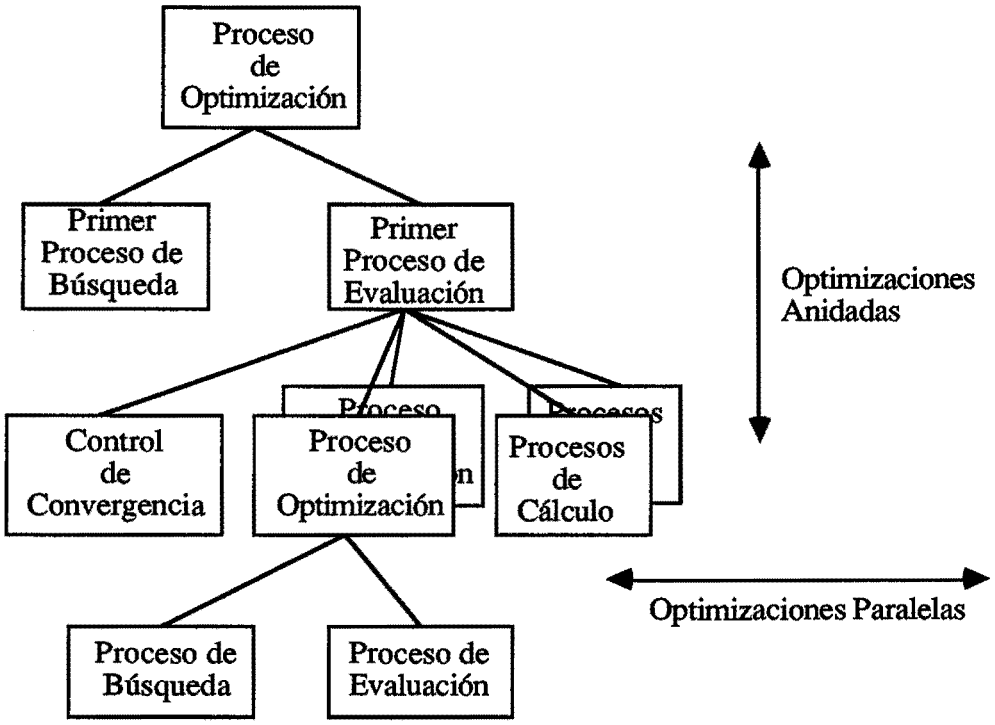


Figura 4.6. Estructura de un Sistema de Optimizaciones Parciales

A su vez, el primer proceso de evaluación constará, en un caso general, de ciertos procesos de cálculo, ciertos subprocesos de optimización y un control de convergencia:

- Los **procesos de optimización** tendrán un cierto número de grados de libertad, correspondientes a otras tantas variables independientes de jerarquía inferior a las del primer proceso de búsqueda. El conocimiento específico de los expertos es crítico para disminuir el número de puntos que es necesario evaluar tanto en el primer proceso de búsqueda como en los de niveles inferiores, con el consiguiente ahorro de tiempo.

- Los **procesos de cálculo** se distinguen de los de optimización por no tener grados de libertad. Para acelerar los procesos de cálculo pueden emplearse procesos simplificados o bien ábacos de interpolación (Capítulo 7).

- El **control de convergencia** está encargado de fijar el orden en que se ejecutan los procesos de cálculo y los subprocesos de optimización. El conocimiento específico de los expertos es indispensable para lograr una convergencia rápida y segura en todos los casos.

Cada subproblema de optimización tiene la misma estructura que el problema inicial de optimización. Tendrá su propio proceso de evaluación y su propio proceso de búsqueda, llevado a cabo en la región de búsqueda del subespacio de sus propias variables independientes.

Los procesos de búsqueda de los subproblemas de optimización (que forman parte del primer proceso de evaluación) son entre sí optimizaciones paralelas, pero son optimizaciones anidadas con respecto al primer proceso de búsqueda.

El proceso de evaluación de cada subproceso de optimización tendrá la misma estructura que el proceso de evaluación del proceso original, es decir, tendrá procesos de cálculo, subprocesos de optimización y control de convergencia.

La "inteligencia" del proceso conjunto de optimización queda así descentralizada, repartida entre un cierto número de "módulos expertos", cada uno de los cuales se encarga de un subproceso (de optimización o de convergencia). Esto permite no sólo enfrentarse siempre a problemas matemáticamente bien condicionados (ver apartado 4.3.4) sino capturar con más facilidad el conocimiento de los expertos, pues se enfrentan a problemas más restringidos. El número reducido de variables en cada subproblema facilita también la implantación de sistemas de aprendizaje automático basados en la sensibilidad de atributos y restricciones respecto a las variables.

Nota bibliográfica:
En muchas publicaciones se reconoce la utilidad de descentralizar el conocimiento en un problema complejo. En [Gupta, 88] y [Kuh & Ohtsuki, 90], por ejemplo, se reconoce la utilidad de jerarquizar las variables independientes y descomponer los procesos tanto en fases como en subproblemas. En [Appelbaum et al, 87] y [Jazdzynski, 89] se descomponen problemas complejos en subproblemas de optimización.
Un ejemplo muy claro de descomposición sistemática en subproblemas es la estructura informática de pizarra (blackboard), que simula un grupo de expertos que plantean y resuelven subproblemas sobre una base de datos común (una pizarra) y se comunican sólo a través de ella. Esta estructura se usa en [Hayes-Roth, 85], [Barbuceanu, 85], [Rehak et al, 85], [Tommelein et al, 87] y [Gupta, 88].

Los "módulos inteligentes" o módulos expertos se llaman specialists en [Brown & Chandrasekaran, 85] y [Harrison et al, 90]; la convergencia se plantea como constraint propagation en [De Kleer & Sussman, 80] y [Dietterich & Ullman, 87]; en [Kalay et al, 87] se habla de planning strategy (experto en convergencia), goals (subproblemas) y evaluators (módulos de cálculo). En [Appelbaum et al, 87] el experto en convergencia se llama coordinator; en [Galiana & McGillis, 88] los módulos de cálculo se denominan mathematical simulation models. Los subproblemas de optimización se llaman design goals en, por ejemplo, [Kalay et al, 87]. En [Shukla, 88] los subproblemas se llaman goals y subgoals, y se organizan secuencialmente según plans y subplans. En [Benders, 62] se habla de problema maestro y problema esclavo (o subproblema).
Dos tipos de descomposición en subproblemas de optimización basados en criterios puramente matemáticos son la descomposición de Benders [Benders, 62] y la Outer-Approximation [Duran & Grossman, 86].

La figura 4.7 muestra la estructura de optimizaciones parciales de la fase 2 de diseño (nivel más detallado) del caso ejemplo de la reactancia, descrito ya en el apartado 3.2. Por ser un caso muy sencillo hay sólo dos optimizaciones parciales anidadas, no hay optimizaciones parciales paralelas y los procesos de cálculo tienen un orden fijo (ver figura 3.8). Por todo ello no son necesarios controles de convergencia.

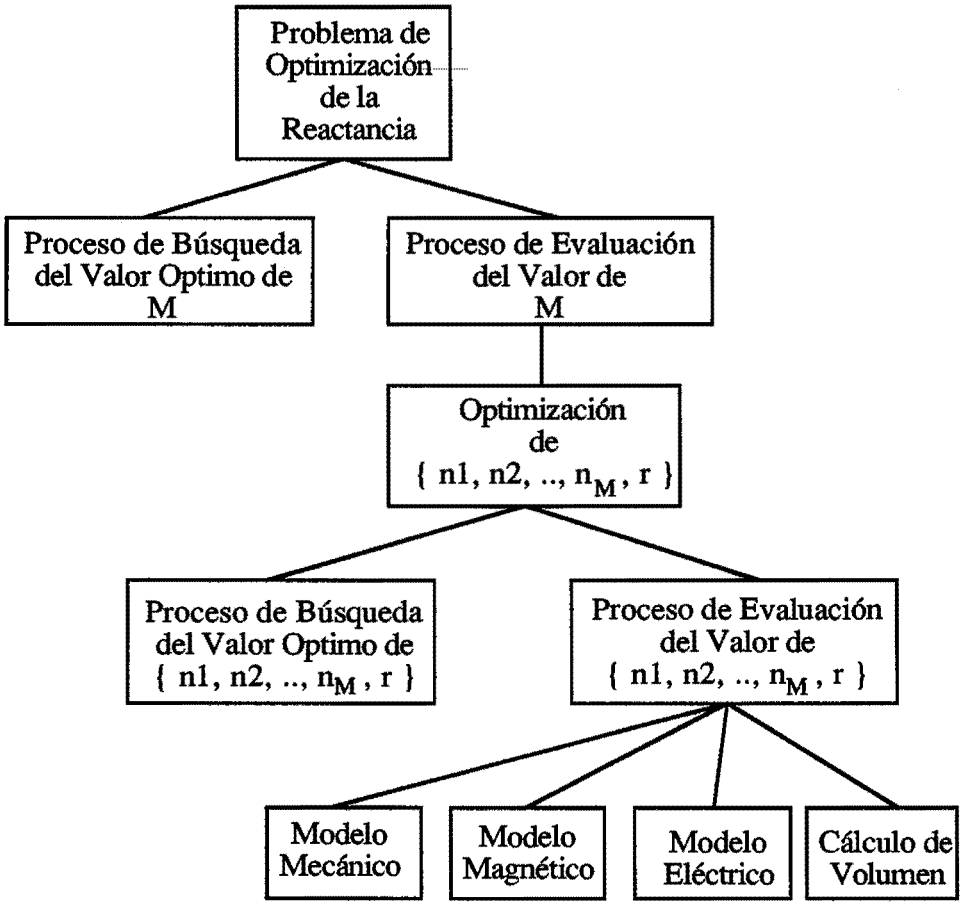


Figura 4.7. Estructura de Optimizaciones Parciales en el Ejemplo de la Reactancia

4.3.4 - Objetivos de una Descomposición en Optimizaciones Parciales.

Cuando una optimización se descompone en optimizaciones parciales cada una de ellas se desarrolla en un subespacio de menor número de dimensiones. Además de esto, la familia de funciones objetivo parciales correspondientes a cada optimización parcial tiene una forma más suave que la función objetivo global, por ser todos sus puntos óptimos parciales de las optimizaciones de nivel inferior (si son anidadas) o de igual nivel (si son paralelas).

Otra ventaja importante es que el conocimiento de los expertos se podrá aplicar con más facilidad y precisión, puesto que los subproblemas (optimizaciones parciales) son mucho más restringidos que el problema total, es decir, tienen menos grados de libertad. Este conocimiento será utilizado tanto para desarrollar métodos que hallen buenos puntos iniciales en las optimizaciones parciales como para desarrollar estrategias de búsqueda eficaces en los distintos subespacios.

Del caso de diseño de la reactancia descrito en el apartado 3.2 se puede extraer un ejemplo muy sencillo del conocimiento experto aplicado a estrategias de búsqueda: si el coeficiente de autoinducción es demasiado bajo, no se debe probar a reducir el número de espiras de las bobinas, sino aumentarlo. Una estrategia ciega probaría ambos casos, llegando a la misma conclusión pero evaluando más puntos.

Habrá en un caso general muchas descomposiciones posibles para una misma optimización y no todas serán igualmente adecuadas. Para que una descomposición determinada resulte ventajosa deberá reunir ciertas **condiciones**:

- Las variables independientes pertenecientes a cada optimización parcial definirán un subespacio vectorial de **dimensión constante** durante todo el desarrollo de dicha optimización parcial.

- Todas las variables independientes pertenecientes a cada optimización parcial serán de **naturaleza matemática homogénea**, es decir: o todas continuas, o todas discretas o todas muy discretas.

- Todas las variables independientes pertenecientes a cada optimización parcial serán **comparables** entre sí en cuanto a su **importancia relativa** respecto a la función

objetivo parcial utilizada en la optimización. Esto significa que la sensibilidad de dicha función respecto a incrementos equivalentes de las distintas variables será del mismo orden de magnitud.

- Podrán desarrollarse buenos métodos de **hallar puntos iniciales** para todas y cada una de las optimizaciones parciales resultantes de la descomposición. Estos métodos tendrán en cuenta tanto el valor de los requisitos del problema como el valor de las variables independientes pertenecientes a optimizaciones parciales de nivel superior (si son anidadas) o del mismo nivel (si son paralelas). Recuérdese que dichas variables se mantendrán constantes en el contexto de la optimización parcial en curso, considerándose parámetros que afectan al espacio de búsqueda y a la propia definición de la familia de funciones objetivo parciales (ver apartados 4.3.1, 4.3.2).

Realizar una descomposición adecuada requiere, como puede apreciarse, un conocimiento suficientemente profundo del problema de optimización planteado, y especialmente de la naturaleza de las distintas variables independientes y de su influencia en el estado de las restricciones y de los atributos. La adquisición de este conocimiento con ayuda de documentación y de entrevistas con expertos es vital para el desarrollo de un algoritmo eficaz que resuelva el problema.

Si la descomposición del problema de optimización cumple las condiciones aquí enumeradas es muy probable que las optimizaciones parciales así planteadas presenten las siguientes **propiedades**:

- Las **regiones factibles** en los subespacios correspondientes a las optimizaciones parciales son conjuntos conexos (posiblemente convexos) y además son fácilmente **localizables** por los procedimientos encargados de generar buenos puntos iniciales.

- En las regiones de búsqueda seleccionadas toda función objetivo parcial presenta un **comportamiento local** deseable, esto es, no presenta más de un mínimo local y se pueden emplear medidas de sensibilidad (como pueda ser el gradiente) para conducir la búsqueda a dicho óptimo.

Si las optimizaciones parciales presentan estas propiedades se puede afirmar que el proceso global de optimización será capaz de explorar **exhaustivamente** y en un **plazo de tiempo razonable** la región de puntos admisibles del problema, hallando todos los **mínimos locales de interés**.

4.4 - Notas sobre Aplicaciones a Proyectos.

Las ideas y metodologías expuestas en este capítulo están inspiradas en el trabajo llevado a cabo en proyectos reales de investigación y desarrollo. En este apartado se comenta el trabajo realizado en dichos proyectos a la luz de las ideas y la nomenclatura utilizadas aquí, como si éstos fueran aplicaciones de la tesis. De esta forma las aplicaciones reales proporcionan un respaldo experimental a las ideas y metodologías expuestas en la tesis. Con este objeto el orden de las notas no es cronológico, sino que se adapta al orden de la exposición del capítulo.

En [Cuadra et al, 85] y [García et al, 87] (ver Apéndice B) se estableció una jerarquía de tres niveles para las variables independientes. Los criterios empleados en la jerarquización fueron la necesidad de llevar a cabo todas las búsquedas en espacios de dimensión constante (por medio de optimizaciones anidadas, ver apartado 4.3) y el "tamaño" de las variables independientes (muy discretas, discretas y continuas; ver apartado 5.1).

La dimensión del espacio de variables del segundo nivel dependía del valor de las variables del primer nivel; a su vez, la dimensión del espacio de variables del tercer nivel dependía del valor de las variables de los dos niveles superiores.

El comportamiento local de las variables era bueno en todos los niveles. En cuanto al tamaño de las variables (número de valores distintos en la práctica), en el primer nivel era 5, en el segundo era 15 y en el tercero oscilaba entre 200 y 1000, según las distintas variables del espacio.

El algoritmo constaba de tres fases de diseño correspondientes a otros tantos niveles de detalle en el modelado. La primera fase de diseño estaba compuesta por dos optimizaciones parciales anidadas, la de nivel superior con variables muy discretas y la de nivel inferior con variables discretas y continuas; la segunda fase estaba compuesta también por dos optimizaciones parciales anidadas, la superior con variables muy discretas y la inferior con variables continuas; la tercera fase de diseño estaba compuesta por dos optimizaciones parciales paralelas, ambas con variables continuas.

En la tercera fase la convergencia de las optimizaciones paralelas era segura, porque el resultado de una de las optimizaciones influía muy poco en el de la otra.

Los resultados obtenidos tanto en [Cuadra et al, 85] como en [García et al, 87] demostraron que las descomposiciones de ambos problemas en optimizaciones parciales fueron adecuadas. No se detectaron problemas de mínimos locales, y el tiempo consumido en la obtención de soluciones variaba consistentemente con la complejidad de las soluciones obtenidas (más bobinas en paralelo, más variables independientes) en función de los requisitos del usuario.

El problema general estudiado en [IIT, 87] era de una complejidad abordable sólo mediante una descomposición estructurada en fases de diseño y optimizaciones parciales. Se estudió concretamente la descomposición del problema de diseño de satélites de comunicaciones, estableciéndose una jerarquía de subsistemas y procesos de cálculo inspirada en la información disponible. El estudio demostró la necesidad de una comunicación clara y precisa con los expertos en el tema para lograr resultados prácticos.

En el prototipo se implantó una sola fase de diseño descompuesta en dos optimizaciones parciales anidadas; la de nivel superior correspondía a variables de tamaño pequeño (entre 2 y 10) y de mal comportamiento local, mientras que la inferior se llevaba a cabo en un espacio de variables de tamaño grande (entre 100 y 1000) y de buen comportamiento local.

En [Cuadra & León, 90] (distribución óptima de objetos en un cajón) se plantearon dos fases de diseño, ambas con la misma estructura de búsqueda en árbol en profundidad. En cada nivel de ramificación del árbol se plantea el subproblema de fijar tres variables jerarquizadas: 1) qué objeto se coloca a continuación, 2) (para cada objeto) en qué orientación, 3) (para cada objeto ya orientado) en qué posición. Cada terna de variables generada {objeto, orientación, posición} producía un nodo "hijo", y todos los nodos hijos de un mismo nodo padre se sometían a una selección multiatributo para expandir sólo los mejores.

La jerarquía de las variables se debe a criterios de importancia relativa respecto al resultado de la terna {objeto,orientación,posición}; en efecto, una vez tomado el objeto, el número de orientaciones posibles está limitado por sus propias restricciones; una vez tomado un par {objeto,orientación}, el número de posiciones queda limitado por los espacios libres en el cajón. Los resultados demostraron que el método seguido de descomposición en fases y subproblemas era adecuado.

En [Cuadra et al, 90] (diseño integrado de motores y equipo electrónico) se estudió la fase de diseño más detallada y compleja, con todas las variables independientes por fijar. La descomposición en optimizaciones parciales alcanzó cuatro niveles; las optimizaciones principales eran dos paralelas (diseño del motor y diseño del inversor), ambas anidadas con respecto a la optimización principal; esta última estaba encargada de las variables más discretas y/o las que afectaban conjuntamente a motor e inversor.

Capítulo 5

Estrategias de Búsqueda

En el Capítulo 3 se mostró cómo un problema complejo de optimización de diseño se puede abordar mediante una serie de fases de optimización sucesivas, en las que el nivel de detalle del modelo del sistema aumenta a medida que se van acotando más las regiones de búsqueda.

El Capítulo 4 describe cómo cada fase de diseño es en general un problema complejo de optimización que puede resolverse eficazmente mediante su descomposición en un sistema de optimizaciones parciales, tanto anidadas como paralelas. También se indica que, si los criterios de descomposición son adecuados a la naturaleza del problema, cada optimización parcial desarrolla un proceso de búsqueda en un espacio de variables independientes homogéneas y de dimensión constante y reducida.

Cada proceso de búsqueda será más eficaz cuanto mejores sean sus estrategias de búsqueda. Una **estrategia de búsqueda** es el razonamiento que guía el proceso de búsqueda, y debe ser capaz de resolver los siguientes problemas:

- Conociendo los últimos pasos seguidos en la búsqueda (los últimos puntos evaluados y las características del diseño resultantes de dichas evaluaciones) debe **proponer un nuevo punto** para ser evaluado.

- Conociendo los últimos pasos seguidos, debe poder **reconocer** cuándo se ha alcanzado un **óptimo local**, dando la búsqueda por terminada.

Un mismo proceso de búsqueda puede utilizar varias estrategias distintas para obtener mejores resultados, incluso utilizando la misma función objetivo y el mismo punto inicial. En una búsqueda local las estrategias se emplearán secuencialmente, mientras que en una búsqueda en árbol se pueden emplear simultáneamente.

La **eficacia** de una estrategia, con respecto a un problema de búsqueda determinado, se puede medir por el número de puntos que necesita evaluar para encontrar la solución del problema. Esta solución podrá consistir en hallar uno o varios óptimos locales, dependiendo de la naturaleza y circunstancias del problema de búsqueda (función objetivo, región de búsqueda, etc.)

No es posible hablar de una eficacia absoluta, o sea, independiente del problema de búsqueda planteado. Un intento de generalización exige una clasificación de los problemas en función de sus características matemáticas y de ciertos parámetros adicionales relacionados con problemas prácticos de implantación informática. De hecho, los problemas puramente prácticos a menudo exigen desarrollos teóricos especiales; por ejemplo, el manejo de grandes matrices vacías constituye un campo de estudio aparte, sin que teóricamente haya diferencia con el manejo de matrices pequeñas.

Un excelente trabajo en la generalización de problemas de búsqueda y optimización que hace énfasis en métodos tradicionales de programación matemática es [Gill et al, 81]. En él (como en la mayoría de los tratados de optimización matemática) se ofrece una clasificación exhaustiva de los problemas de optimización según las características de la función objetivo, las restricciones, la naturaleza y número de las variables independientes, etc. También incluye una clasificación de las estrategias de búsqueda según el tipo de problema, y muchas consideraciones prácticas con vistas a la implantación eficaz de algoritmos.

Como norma de trabajo se recomienda hacer uso del conocimiento de los expertos, ya sea de tipo heurístico o bien de tipo teórico, con lo que la estrategia se puede desarrollar (o adaptar) a la medida del problema para hacerla más eficaz. Hay sin embargo algunas características generales de las variables independientes que pueden emplearse para hacer más eficaces las búsquedas en un problema genérico.

Una característica básica de cada optimización parcial es la **naturaleza matemática** de sus variables independientes, que permite clasificarlas en grupos como variables **continuas**, variables **discretas** y variables **muy discretas.**

En rigor sólo debería distinguirse entre variables continuas, asociadas a conjuntos densos de números (como los números reales y los números racionales), y variables discretas, asociadas a conjuntos no densos de números (como los enteros y los naturales); pero la práctica sugiere, por ejemplo, no considerar como pertenecientes al mismo grupo a una variable discreta que admite sólo dos valores y a otra también discreta que admite cien valores distintos.

En el apartado 5.1 se proponen dos criterios prácticos para clasificar las variables independientes: el tamaño y el comportamiento local. Según estos criterios se definen dos prototipos de estrategias de búsqueda, el de variables continuas y el de variables muy discretas.

El apartado 5.2 está dedicado a las estrategias para variables continuas, describiéndose con detalle una de ellas (algoritmo de Hooke & Jeeves) por haber sido utilizada ampliamente en los proyectos que respaldan esta tesis.

El apartado 5.3 está dedicado a las estrategias para variables muy discretas. En él se distinguen dos grupos: las estrategias que abordan la búsqueda discreta como tal (llamadas aquí de forma general búsquedas en árbol) y las estrategias que transforman el problema discreto en continuo para facilitar su resolución. De estas últimas se describen con detalle el Etiquetado Consistente, por haber sido ensayado realmente en proyectos (aunque no comerciales), y la búsqueda en Espacio Heurístico, por ser una formulación original de esta tesis.

Finalmente, el apartado 5.4 está dedicado a comentar aspectos relacionados con este capítulo de los proyectos reales de diseño por ordenador que han dado lugar a esta tesis.

5.1 - Criterios para la Clasificación de Variables Independientes.

La forma de clasificación de las variables que aquí se propone se basa en criterios de origen práctico que a veces no será fácil estimar. Estos criterios son el **tamaño** de las variables y su **comportamiento local**; el primero es un criterio cuantitativo y el segundo es cualitativo.

El **tamaño** de una variable, en el contexto de un problema dado, se define aquí como el número máximo de valores distintos que puede tomar la variable durante el proceso de resolución de dicho problema. El tamaño de una variable coincide con el concepto matemático

de "cardinal" de un conjunto, coincidencia lógica si se piensa que cualquier conjunto se puede asociar a una variable vectorial.

Nota bibliográfica:
En [Wirth, 80] se define el concepto de "tamaño" como cardinalidad, visto desde las necesidades de memoria de una variable:

> "El número de valores distintos que pertenecen al tipo T se llama cardinalidad (o cardinal) de T. La cardinalidad proporciona una medida de la cantidad de memoria que se necesita para representar una variable."

En la literatura técnica sobre optimización los problemas se clasifican, entre otros criterios, por la naturaleza matemática de las variables que intervienen (no sólo de las independientes, pues ya se ha indicado que las ligaduras se suelen tratar como restricciones de igualdad); así se distingue entre variables enteras (cuando son discretas) o reales, y en especial booleanas (boolean variables) cuando son discretas y sólo admiten dos valores distintos.
En [Brown & Chandrasekaran, 85] las variables discretas se llaman alternatives; en [Dietterich & Ullman, 87] se distingue entre structural decisions y variables; decisions en [Shukla, 88]; critical decisions en[Zhang, 89]; en [Ishii & Barkan, 87] se distinguen choices y variables; en [Arbab, 89] las variables discretas se denominan alternatives y las continuas attributes; en [Grossman, 90] se tratan por separado las discrete decisions y las continuous variables.

El hecho de condicionar la definición de tamaño al contexto de cada problema se debe a la orientación práctica de la definición; en general se manejan las variables con una precisión limitada, que se expresa como un número máximo de cifras significativas, unos incrementos mínimos, unos criterios de redondeo, etc. Esto significa que en la práctica todas las variables se consideran discretas. Por otro lado, aunque las regiones de búsqueda puedan ser teóricamente infinitas, en la práctica estarán acotadas por límites razonables. Al ser las variables discretas y estar acotadas las regiones de búsqueda, cada región de búsqueda tendrá un número finito de puntos que se puede calcular a partir del tamaño de las variables independientes.

El tamaño mínimo que puede tener una variable es 2, correspondiendo a una decisión elemental (0/1, Verdadero/Falso, etc...) típica del álgebra de Boole. No existe un límite teórico para el tamaño máximo, pero en aplicaciones reales son altos los valores de 10^3 y 10^4. Por ejemplo, si la intensidad de corriente de un motor eléctrico puede variar entre 0 y 100 A y se considera un incremento mínimo de 0.1 A, entonces el tamaño de la variable será de 10^3.

Una aplicación interesante del tamaño limitado de las variables es que en muchos problemas todas ellas se podrán representar por números enteros, simplemente tomando como unidad el incremento mínimo correspondiente. Esto permite mejorar la eficacia de las aplicaciones informáticas, especialmente en cuanto a la memoria utilizada y al tiempo de proceso.

El **comportamiento local** de una variable independiente con respecto a un problema dado no es un parámetro tan fácil de cuantificar como el tamaño. Se puede definir como la

posibilidad de realizar con éxito interpolaciones y extrapolaciones con objeto de predecir los resultados de las evaluaciones en zonas todavía no exploradas.

El comportamiento local mide por tanto la capacidad de anticipar los valores de los atributos y restricciones del problema en puntos todavía no evaluados, basándose en las posiciones y características de los puntos evaluados. Está relacionado estrechamente con las propiedades locales de las funciones, como la monotonía, la convexidad, etc. Por ejemplo, el comportamiento local óptimo corresponde a las funciones lineales.

Nota bibliográfica:
El comportamiento local, al igual que el tamaño, es un criterio práctico. Dado el entorno matemático en que se mueve la literatura técnica sobre optimización, en ella se atiende más a criterios rigurosos sobre las funciones que intervienen en el problema: funciones lineales, cuadráticas, polinómicas, suaves (smooth), etc. (véase por ejemplo [Gill et al, 81]).
Una aplicación integral del concepto de comportamiento local es el estudio de la monotonía de las funciones que intervienen en un problema con respecto a las variables. Esta técnica se conoce como monotonicity analysis, y se aplica a la optimización de diseño en [Papalambros & Wilde, 79] y [Agogino & Almgren, 87].

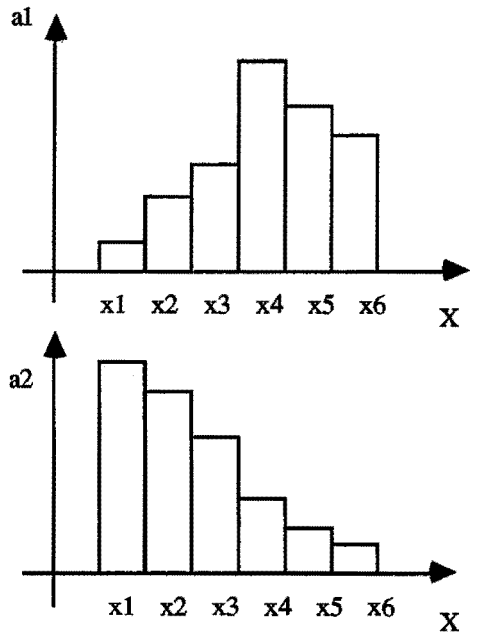


Figura 5.1.a. Variable discreta X de tamaño 6 que presenta un buen comportamiento local con respecto a los atributos a1 y a2.

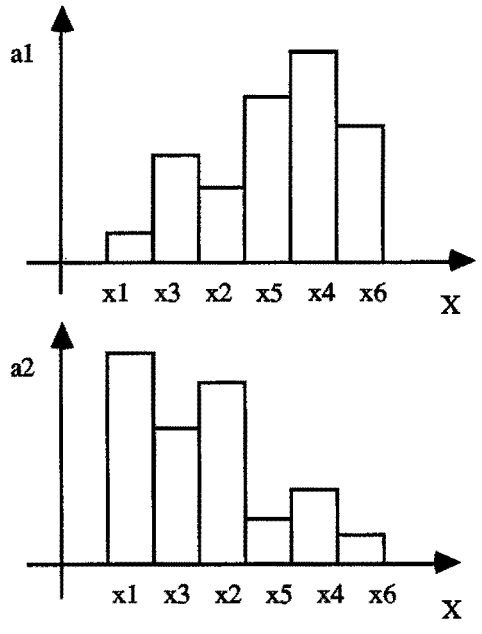


Figura 5.1.b. Una ordenación poco adecuada en los valores de X hace que presente un mal comportamiento local con respecto a los atributos.

El comportamiento local de una variable se define con respecto a un problema dado, y concretamente respecto a los atributos y restricciones de dicho problema; también deben tenerse en cuenta los incrementos mínimos elegidos para cada variable (relacionados con el tamaño de dicha variable). La figura 5.1 trata de ilustrar el concepto de comportamiento local mostrando la influencia de la variación de una variable discreta en el valor de dos atributos diferentes.

La figura 5.1.a muestra una variable discreta tal que es posible realizar interpolaciones y extrapolaciones por su buen comportamiento local con respecto a los atributos. La figura 5.1.b muestra cómo el comportamiento local puede cambiar según la ordenación de valores de la variable (aplicable sólo a variables discretas). Se observa también en la figura 5.1 que el concepto de "**local**" está íntimamente ligado al **tamaño** de cada variable.

5.2 - Estrategias para Variables Continuas y Discretas.

Este apartado trata de las estrategias de **búsqueda local**, adecuadas para variables independientes que se caractericen por tener un **tamaño grande** y un **buen comportamiento local**, propiedades generalmente asociadas a las variables **continuas**. En el caso de que una variable discreta presente estas características se podrá manejar como una variable continua, utilizando las estrategias de búsqueda propias de estas variables.

En [Gill et al, 81] se presenta una amplia gama de estrategias de búsqueda local, adecuadas a distintos tipos de problemas. Un criterio fundamental para clasificar los problemas es el tipo de las funciones (lineales, cuadráticas, etc.) que proporcionan el estado de las restricciones y el valor de los atributos; otros criterios son el tipo de las variables (enteras, reales, etc.) y las características de la región factible (conjunto conexo, conjunto convexo, véase [Borrel, 87]).

El hecho de que las expresiones de las restricciones, atributos y funciones objetivo sean **lineales** representa un caso extremo de buen comportamiento local, pues permite extrapolar e interpolar con total exactitud. En estos problemas se pueden aplicar estrategias muy especiales, que no se tratarán aquí por ser ampliamente conocidas y porque los problemas lineales aparecen muy pocas veces en el contexto de la optimización de diseño.

El buen comportamiento local se aprovecha para explorar grandes regiones de búsqueda (variables de gran tamaño) y encontrar óptimos locales evaluando un número lo más pequeño

posible de soluciones. Hay métodos basados en el gradiente de las funciones (restricciones, atributos), y en general en el valor de las derivadas primera, segunda, etc...que hacen uso de la extrapolación para guiar la optimización hacia zonas potencialmente mejores. Hay también métodos que, basándose en la generación aleatoria de soluciones en la región de búsqueda, utilizan la interpolación para aproximarse a los óptimos locales.

Las derivadas de las expresiones de restricciones, atributos y funciones objetivo normalmente no pueden calcularse explícitamente, debiendo ser aproximadas mediante la evaluación de puntos muy próximos. Esto hace que los métodos basados en los valores de las derivadas se parezcan mucho a los métodos llamados de **búsqueda directa** (direct search en [Hooke & Jeeves, 61]).

Nota bibliográfica:
En [Gill et al, 81] se describen numerosas estrategias para variables continuas y muchas de sus variantes. En [Balas & Martin, 80] se propone una estrategia llamada "pivot and complement" que incluye búsquedas locales en espacios de variables muy discretas (0-1).
En lo que respecta al problema específico del diseño, en [Appelbaum et al, 87] y [Fei et al, 89] se emplea una estrategia llamada "boundary search along active constraints"; también en [Fei et al, 89] se emplea el "Han-Powell method"; en [Singh et al, 83) se emplea el "SUMT algorithm" y el "Rosenbrock's method" (este último es una variación del método de Hooke & Jeeves).

A continuación se describe un método de búsqueda directa que ha sido utilizado con éxito en varios proyectos estrechamente relacionados con el desarrollo de esta tesis, como [Pérez-Arriaga, 78], [Cuadra et al, 85], [IIT, 87] y [García et al, 87], tanto con variables continuas como con variables discretas de buen comportamiento local. Es el algoritmo de **Hooke & Jeeves** (o también estrategia, según la nomenclatura usada aquí), descrito por primera vez en [Hooke & Jeeves, 61].

5.2.1 - El Algoritmo de Hooke & Jeeves.

Este algoritmo ha sido seleccionado en los proyectos ya mencionados y se expone en esta tesis por su sencillez y generalidad. Al ser un algoritmo de búsqueda directa no necesita que las funciones evaluadas sean derivables, y trabaja tanto con variables discretas como con variables continuas; de hecho, todas las variables se consideran discretas al definir sus incrementos mínimos.

El algoritmo de Hooke & Jeeves es de aplicación general a búsquedas locales, aunque se puede mejorar su eficacia mediante decisiones basadas en el conocimiento propio de cada problema de optimización particular. El uso del conocimiento experto de tipo causa-efecto

puede reducir considerablemente el número de puntos evaluados. Este conocimiento permite prever con un error aceptable el efecto que tendrá sobre el estado de restricciones y atributos un conjunto de cambios en las variables independientes.

La estrategia de Hooke & Jeeves consiste en varias etapas de búsqueda con un incremento constante para cada variable independiente (pero distinto para cada una), reduciéndose tras cada una de dichas etapas el valor de los incrementos para refinar progresivamente la solución. Será necesario por tanto especificar el **número de etapas**, el **incremento mínimo** de cada variable y un **criterio de reducción de incrementos** entre dos etapas consecutivas (por ejemplo, dividir por una constante el incremento anterior).

La figura 5.2 muestra una etapa de este algoritmo, en la que ΔX son los incrementos correspondientes a la etapa.

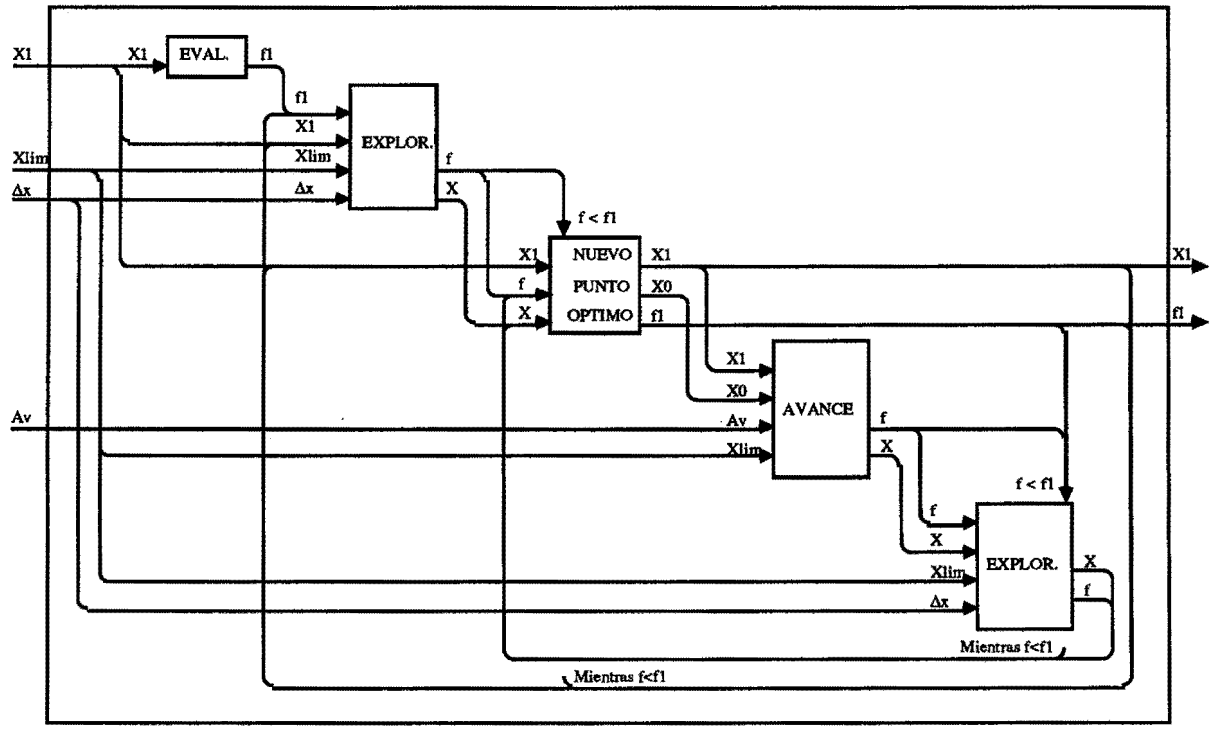


Figura 5.2. Una etapa del algoritmo de Hooke & Jeeves para minimizar la función f.

La optimización es monoatributo, es decir, se considera una sola función objetivo f(X), cuyo valor aparece en la figura como f o como f_1, según sea la variable X o X_1. La región de búsqueda se define por medio de la variable vectorial X_{lim}, que acota los valores de la variable vectorial independiente X entre unos límites inferiores y superiores. X es el valor provisional

de las variables independientes, X_1 es el mejor punto encontrado hasta el momento (óptimo local provisional) y X_0 es el último óptimo local encontrado, es decir, el que ha sido reemplazado por X_1.

El proceso de optimización parte de un **punto inicial**, que por supuesto también es el óptimo provisional al principio del proceso y por eso aparece en la figura como X_1. La búsqueda consiste en una secuencia de movimientos de **exploración** y movimientos de **avance**, guiados por los resultados del **proceso de evaluación**, esto es, por los valores de la función objetivo.

Un movimiento de **exploración** se lleva a cabo incrementando secuencialmente cada variable y comprobando si el nuevo punto es mejor que el anterior, según se muestra en la figura 5.3. Naturalmente no se deberá salir de la región de búsqueda, para lo cual se tiene en cuenta la variable X_{lim}. En caso de que un punto caiga fuera de dicha región no se evalúa, sino que simplemente se rechaza como si fuera peor.

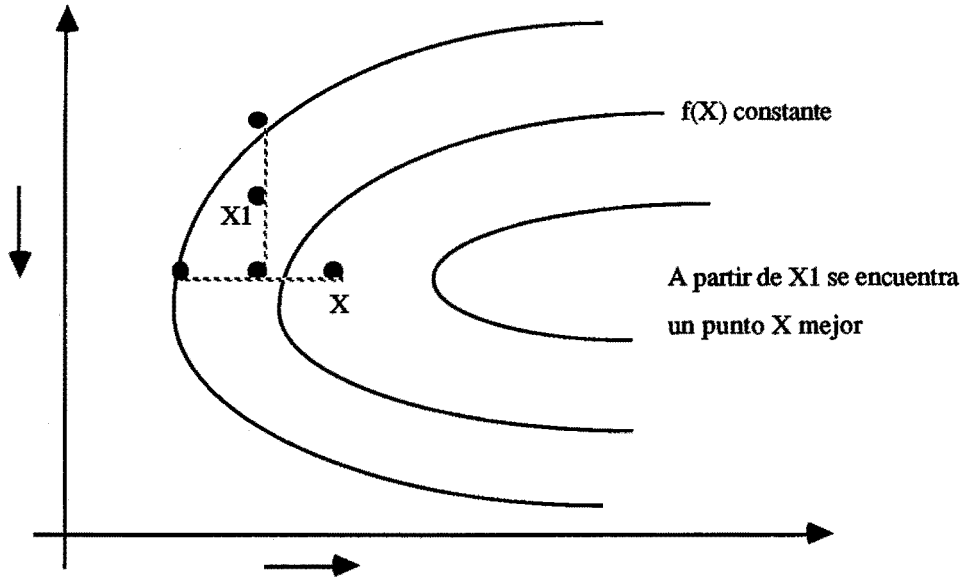


Figura 5.3. Movimiento de Exploración.

Un movimiento de exploración consta de los siguientes pasos:

i) Se selecciona una variable.
ii) Se evalúa el resultado de un **incremento positivo** en dicha variable:

- Si es mejor, se deja la variable incrementada y se vuelve a (i).
- Si es peor, se pasa a (iii).

iii) Se evalúa el resultado de un **incremento negativo** en dicha variable:

- Si es mejor, se deja la variable decrementada y se vuelve a (i).
- Si es peor, se recupera el punto original y se vuelve a (i).

Un movimiento de exploración tiene **éxito** si se obtiene un punto mejor que el de partida; en este caso el punto X se convierte en X_1 y el antiguo X_1 se convierte en X_0. Si **fracasa**, el punto de partida es un óptimo local según el tamaño actual de los incrementos y según la lógica seguida en la exploración (recuérdese que la condición de óptimo local está íntimamente ligada a la estrategia seguida). En cuanto se encuentra un óptimo local, se da por finalizada la etapa de búsqueda y se utiliza dicho óptimo local como punto inicial de la siguiente etapa.

Un movimiento de **avance** trata de ahorrar evaluaciones de puntos dando un paso en la aparentemente mejor dirección, que será la de máxima pendiente según la información disponible. Esta información consiste en los valores de las variables independientes en los puntos X_0 y X_1, que fijan una dirección de avance y un sentido (de X_0 a X_1, es decir de peor a mejor). El módulo del avance se puede modificar mediante un factor de avance, que en la figura 5.2 aparece como Av. El punto de llegada del avance se halla por:

$$X = X_1 + Av\ (X_1 - X_0)$$

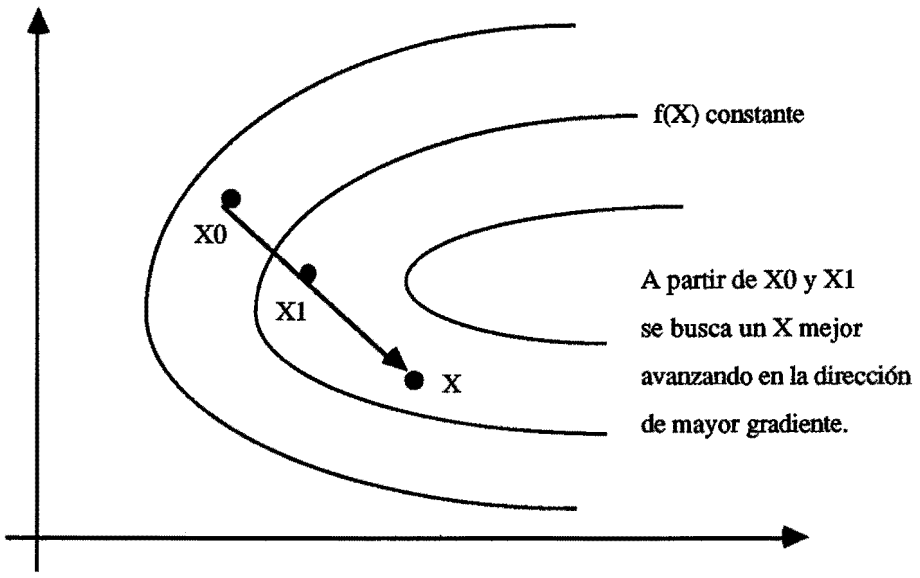


Figura 5.4. Movimiento de Avance.

La figura 5.4 muestra el movimiento de avance que seguiría a la exploración de la figura 5.3. Con el avance se modifican los valores de los componentes de X, y hay que controlar que no se salgan de la región de búsqueda. Si alguna variable se sale de la región, su valor se "satura": adquiere el valor límite que corresponda según se haya salido por el límite inferior o por el superior. Esto último es una adaptación particular del algoritmo.

En el punto de llegada del movimiento de avance se realiza un movimiento de **exploración**, obteniendo un punto provisional X mejor o igual que el punto de llegada. Si X es peor que X_1 ($f_1 < f$) el avance **fracasa** y se vuelve a X_1, donde se realizará una nueva **exploración**. Si por el contrario X es mejor que X_1 ($f < f_1$), el avance es un **éxito**, y el punto X se convierte en X_1 y el antiguo X_1 se convierte en X_0, para realizar a continuación otro movimiento de **avance**.

5.3 - Estrategias para Variables Discretas y Muy Discretas.

Este apartado está dedicado a las estrategias usadas para optimizar el valor de variables de **tamaño pequeño** (pocos valores distintos posibles) y **mal comportamiento local** (no permiten interpolar ni extrapolar resultados).

Se considera como variables **muy discretas** a aquellas que admiten muy pocos valores diferentes y además un cambio en dichos valores provoca una variación muy significativa en las características del sistema que se desea optimizar.

Las optimizaciones parciales definidas en espacios de variables independientes discretas y muy discretas presentan **dificultades** especiales:

- Suelen ser las variables más importantes, por lo que acostumbran a estar en los niveles más altos de las jerarquizaciones de variables. Esto hace que el **tiempo** necesario para **evaluar** con detalle cada punto sea muy alto, por incluir dicha evaluación las optimizaciones parciales de nivel inferior (ver apartado 4.3.1, Optimizaciones Anidadas).

- Aunque el tamaño de cada variable sea pequeño por separado, el **número** total de puntos generado por la combinación de varias de ellas puede llegar a ser muy alto.

- Al no poder extrapolar ni interpolar resultados, ni siquiera para cada variable por separado, **deberá evaluarse la mayoría** de los puntos posibles del problema.

- Es frecuente que las variables muy discretas de una optimización parcial estén **fuertemente ligadas** entre sí, lo que hace que los cambios individuales en el valor de las variables no aporten información sobre el efecto de los posibles cambios conjuntos.

Para hacer frente a estas dificultades se deben utilizar algunas de las **características** favorables de estas variables:

- Dado que suelen ser variables muy importantes, se puede evaluar su influencia con **modelos poco precisos** de la realidad. De este modo será posible evaluar un gran número de puntos en un tiempo razonable, teniendo en cuenta siempre los márgenes de error correspondientes.

- También debido a su importancia, los expertos suelen **conocer** bien la **influencia** de estas variables en el sistema (diseño) que se desea optimizar, al menos a nivel cualitativo. Esto permitirá reducir significativamente el tamaño real de la región de búsqueda.

- Dado que en la optimización de variables discretas no se puede explorar apoyándose en el comportamiento local, es igualmente complicado realizar una optimización monoatributo o una **optimización multiatributo**. Esto es así porque la tarea no consiste normalmente en "generar soluciones mejores" (como en las estrategias propias de variables continuas) sino en "eliminar soluciones peores", y esto último se puede llevar a cabo fácilmente con uno o varios criterios de selección, como se verá en el Capítulo 6.

La primera característica, es decir, el uso de modelos simplificados para seleccionar los valores más interesantes de las variables muy discretas, está muy relacionada con la descomposición de una optimización en varias fases (apartado 3.4), cada fase con una región de búsqueda más limitada y un modelo más detallado que en la anterior.

La segunda característica apoya la necesidad de incorporar el conocimiento propio del problema a las estrategias de búsqueda para aumentar su eficacia.

Este apartado está estructurado como sigue:

- El apartado 5.3.1 trata de métodos de resolución que manejan las variables discretas como tales, representables siempre como búsquedas en árbol. En particular se comentan dos tipos de **búsqueda en árbol**: el de **soluciones incompletas** y el de **soluciones completas**. La búsqueda en espacios de variables discretas se llama también búsqueda combinatoria (por ejemplo en [Borrel, 87]).

- Los apartados 5.3.2 y 5.3.3 tratan métodos basados en el cambio de espacio de búsqueda para poder llevar a cabo **búsquedas locales**. El 5.3.2 describe el **etiquetado consistente** (consistent labelling, [Haralick & Saphiro, 79], [Hummel & Zucker, 83]), que puede resolverse trasladando el problema discreto a un espacio continuo, aumentando con ello el número de variables. El 5.3.3 describe el cambio a un **espacio heurístico**, que agrupa las variables discretas en familias según criterios heurísticos y asigna a cada familia una variable casi continua (es decir, discreta pero con propiedades de buen comportamiento local). Este último ha sido desarrollado en el contexto de esta tesis y no ha sido todavía aplicado a proyectos reales.

5.3.1 - Búsquedas en Arbol.

La representación formal más adecuada de un espacio de variables independientes discretas es la de un árbol de decisiones, fijando cada decisión el valor de una o de varias variables independientes del diseño. Se presentan aquí dos tipos de árboles distintos, el de soluciones incompletas y el de soluciones completas:

- El árbol de **soluciones incompletas** se basa en la generación progresiva de soluciones en ausencia de un punto de partida.

- El árbol de **soluciones completas** se basa en la mejora sistemática de soluciones a partir de un punto inicial del espacio de variables independientes.

El árbol de **soluciones incompletas** se forma a partir de un nodo inicial en el que se desconoce el valor de todas las variables independientes. Está estrechamente relacionado con la

búsqueda heurística (ver apartado 2.2), pues para evitar la generación y evaluación de todas las soluciones posibles hay que "podar" ramas del árbol, y por tanto hay que evaluar soluciones incompletas por medio de heurísticos.

Nota bibliográfica:
El clásico algoritmo Branch & Bound [Beale, 77] es un ejemplo de estrategia de búsqueda para variables discretas por árbol de soluciones incompletas, auxiliado por un heurístico especialmente útil para acotar el óptimo obtenido suponiendo que las variables que quedan por fijar fueran continuas.
La búsqueda en árbol de soluciones incompletas es un caso de razonamiento no monótono (lógica presuposicional), pues las decisiones se han de tomar basándose en información incompleta. Por ejemplo en [Waldron, 89] se definen las decisiones monótonas como:
"...the decisions which will not be violated by the decisions yet to be made."
En las búsquedas en árbol el hecho de reconsiderar una decisión ya tomada se suele denominar backtracking.

Este tipo de representación se suele aplicar a problemas dinámicos, esto es, problemas en los cuales el orden en que se fijan las variables independientes es crítico. También es adecuado para resolver problemas donde el espacio de variables independientes es de dimensión variable.

Cada nivel de profundidad del árbol representa una decisión de fijar una o varias variables independientes, mientras que la ramificación representa los valores alternativos que pueden tomar las variables.

Supóngase por ejemplo un problema con tres variables independientes discretas, pudiendo cada una tomar tres valores posibles { a, b, c }; el tamaño de las tres variables es por tanto igual a tres. Un árbol de búsqueda posible de soluciones incompletas sería el mostrado en la figura 5.5:

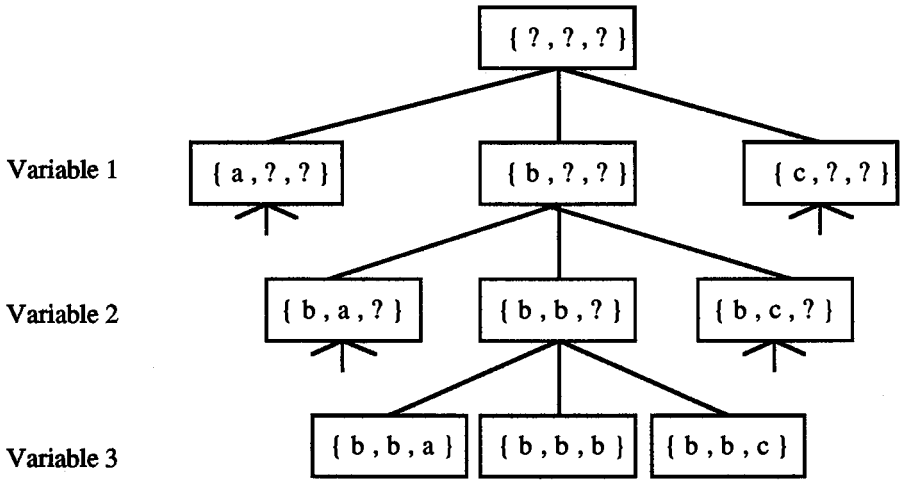


Figura 5.5. Un Arbol de Soluciones Incompletas

Cada nivel de profundidad del árbol representa el hecho de fijar una o varias variables independientes; por tanto se pueden formalizar con facilidad las restricciones que relacionan cada variable con las demás, reduciendo el número de valores alternativos (ramificaciones, nodos hijo) de dicha variable en función de los valores de las variables ya fijadas en el nodo padre. De esta manera todos los nodos del árbol representan soluciones provisionalmente factibles.

Del mismo modo puede aplicarse el conocimiento heurístico de los expertos en cada etapa de expansión, seleccionándose sólo aquellos nodos hijos potencialmente interesantes. También será necesario evaluar los nodos hijos con heurísticos para expandir primero los mejores y rechazar los peores.

Nota bibliográfica:
En el campo de la optimización de diseño los problemas dinámicos no son tan frecuentes como en el campo de la optimización de planes. Algunos problemas de diseño dinámicos que se pueden citar son la acomodación óptima de objetos (subsistemas, componentes) en un espacio dado (ver [Cuadra & León, 90]) , la selección de componentes (como en [Dietterich & Ullman, 87]), el diseño de estructuras y selección de materiales (véase [Maher & Fenves, 85]), etc.
Trabajos de optimización de diseño en los que se llevan a cabo búsquedas en árbol de soluciones incompletas son: [Maher & Fenves, 85], [Kalay et al, 87], [Dietterich & Ullman, 87], [Cuadra & León, 90] y [Navinchandra & Marks, 87].

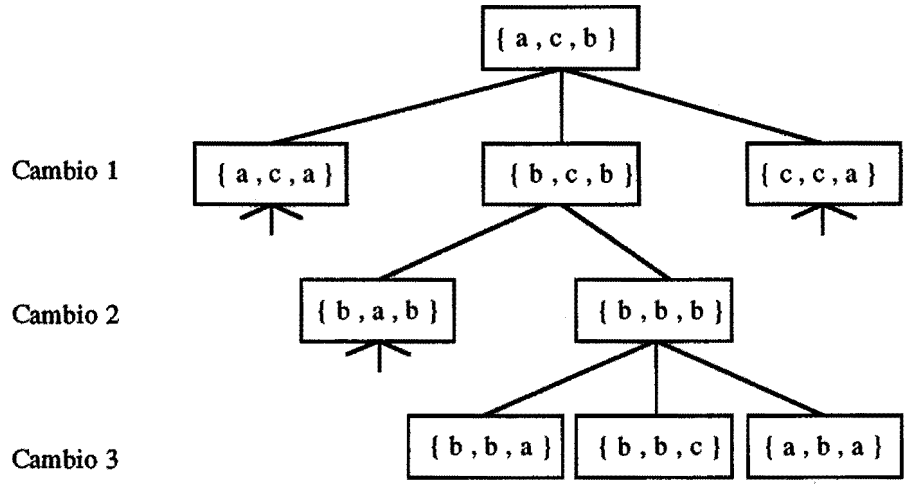


Figura 5.6. Un Arbol de Soluciones Completas

El árbol de **soluciones completas** se crea a partir de un punto inicial de la región de búsqueda, que podrá ser o podrá no ser factible. Por lo tanto se conoce el valor inicial de todas las variables independientes. Esto hace que esta representación sea más adecuada en espacios de dimensión constante, lo que siempre se podrá conseguir por medio de una descomposición en optimizaciones parciales (ver Capítulo 4).

El nivel de profundidad del árbol representa el número de acciones de cambio aplicadas al punto o nodo inicial, mientras que la ramificación representa las acciones de cambio alternativas aplicables al mismo punto o nodo. Al igual que en los árboles de soluciones incompletas, cada acción puede afectar a una o varias variables simultáneamente.

La figura 5.6 muestra un posible árbol de soluciones completas para el mismo caso ejemplo de la figura 5.5.

Cada cambio puede responder a una estrategia de optimización distinta, a una función objetivo distinta o a ambas cosas a la vez. Lo fundamental es que los cambios se basan en evaluaciones reales de restricciones y atributos de los puntos o nodos, no necesitándose heurísticos adicionales porque se conoce el valor de todas las variables independientes.

La búsqueda en árboles de soluciones completas encaja con el tipo de búsqueda llamada **planificación** (ver apartado 2.2.2) pues se basa más en un control cuidadoso en la aplicación de los operadores de cambio que en la evaluación de estados intermedios con heurísticos. Debe tenerse en cuenta que el nivel teórico de ramificación es mucho más grande en este tipo de representación que en la de soluciones incompletas, y además todas las restricciones se deben considerar simultáneamente.

El conocimiento experto de tipo "causa-efecto" es especialmente útil en este tipo de árboles, pues se intentará mejorar la evaluación de cada nodo en cuanto a restricciones y atributos mediante cambios en el valor de las variables independientes.

En principio cada tipo de árbol se ajusta a problemas de distintas características, especialmente en cuanto a las variables independientes y a la forma de las restricciones. Esto no impide que puedan complementarse a la perfección, pues el árbol de soluciones incompletas puede utilizarse para encontrar un conjunto de puntos iniciales factibles e interesantes, para generar luego a partir de cada uno un árbol de soluciones completas. Esta política encaja perfectamente con la descomposición de un problema en fases de diseño (ver apartado 3.4).

Nota bibliográfica:
Trabajos de optimización de diseño en los que se llevan a cabo búsquedas en árbol de soluciones completas son [Tommelein et al, 87], [Murthy & Addanki, 87] y [Latorre et al, 90].

Tanto en un tipo de árbol como en el otro, el hecho de tener que evaluar un gran número de soluciones puede hacer que los tiempos de cálculo sean inadmisibles. En los casos en que el número de combinaciones de las variables muy discretas sea alto, el tiempo de cálculo se puede controlar si:

- Existen criterios para eliminar rápidamente, e incluso ni siquiera llegar a evaluar, soluciones poco apropiadas para los requisitos del sistema (o diseño).

- Existen modelos simplificados que sirven para comparar soluciones entre sí muy rápidamente, teniendo en cuenta un margen razonable de incertidumbre a la hora de eliminar soluciones.

5.3.2 - Etiquetado Consistente.

En este apartado se hace una breve descripción formal de este enfoque, comentando sus características principales. En cuanto a su aplicación al diseño, el etiquetado consistente puede usarse para resolver búsquedas en variables discretas y búsquedas aproximadas en variables continuas previamente discretizadas, según los principios del llamado razonamiento cualitativo (ver apartado 2.2.2).

El etiquetado consistente (consistent labelling, [Haralick & Saphiro, 79], [Hummel & Zucker, 83]) trata de resolver problemas con un gran número de variables discretas convirtiéndolos en problemas de variables continuas; entre los distintos valores de las variables se supone que hay múltiples relaciones de compatibilidad e incompatibilidad que admiten una cierta jerarquía, cuantificable según su importancia relativa.

Nota bibliográfica:
Conviene aclarar que en rigor el Etiquetado Consistente es el planteamiento del problema (nodos, etiquetas y relaciones de compatibilidad) y no el método de resolución por variables continuas descrito aquí. Por ejemplo, en [Navinchandra & Marks, 87] se resuelve un problema de etiquetado consistente mediante una búsqueda en árbol de soluciones incompletas, manteniendo las etiquetas como variables discretas. La función de consistencia global se utiliza en este caso para evaluar los distintos nodos.
En [Freuder, 76] el mismo problema no se denomina etiquetado consistente, sino constraint propagation, y las relaciones de compatibilidad se consideran no relajables, es decir, si hay alguna contradicción el problema se considera insoluble.

A cada variable independiente se le llama **nodo**, y puede tomar un número limitado de valores; se desea averiguar la combinación o combinaciones de valores de los nodos que son compatibles simultáneamente, y posiblemente óptimas con respecto a ciertos criterios. Los distintos valores de cada variable se llaman **etiquetas**.

Sea por ejemplo un problema de tres nodos:

$$X = \{ A, B, C \}$$

Sea m_a el número de etiquetas del nodo A, es decir, el tamaño de la variable A. Supóngase ahora que las etiquetas de los tres nodos son:

$$A = \{ a_0, a_1, a_2 \} \qquad m_a = 3$$
$$B = \{ b_0, b_1 \} \qquad m_b = 2$$
$$C = \{ c_0, c_1 \} \qquad m_c = 2$$

Las relaciones de compatibilidad se formulan como unos coeficientes reales (positivos o negativos, según se trate de compatibilidad o incompatibilidad) asociados a conjuntos de etiquetas de los distintos nodos.

Una relación $Ci[a_j; b_k, c_l]$ representa la compatibilidad de la etiqueta a_j con respecto a las etiquetas b_k y c_l. Por ejemplo, se puede proponer el siguiente grupo de relaciones de compatibilidad:

$$C1[a_0; b_0] = +10$$
$$C2[a_1; b_1, c_0] = -5$$
$$C3[a_1; b_0, c_1] = -15$$
$$C4[b_1; a_2] = +8$$
$$C5[c_0; a_0] = +2$$

Se asigna ahora a cada etiqueta de cada nodo una variable continua que aquí se representará con el mismo nombre de la etiqueta. Estas variables continuas cumplirán para todo nodo (se toma aquí como ejemplo el nodo A) que:

$$0 \leq a_i \leq 1$$

$$\sum_{i=0}^{m_a-1} a_i = 1$$

Se define ahora el **soporte** de una etiqueta de un nodo, con respecto a una relación de compatibilidad en la que dicha etiqueta participa, como el producto del coeficiente de dicha relación por los valores de las variables asociadas a las demás etiquetas participantes en la relación. Por ejemplo, el soporte de la etiqueta a_1 con respecto a la relación C3 sería:

$$S(a_1,C3) = -15\ b_0 c_1$$

El soporte global de una etiqueta de un nodo es la suma de los soportes de dicha etiqueta con respecto a todas sus relaciones de compatibilidad. Por ejemplo, el soporte global de la etiqueta a_1 sería:

$$S(a_1) = S(a_1,C2) + S(a_1,C3) = -5\ b_1 c_0 - 15\ b_0 c_1$$

Se denomina función de **consistencia global** a la suma de los productos del valor de cada etiqueta por su soporte global. Por ejemplo, la función de consistencia global del caso ejemplo sería:

$$S = \sum_{i=0}^{m_a-1} a_i S(a_i) + \sum_{i=0}^{m_b-1} b_i S(b_i) + \sum_{i=0}^{m_c-1} c_i S(c_i)$$

$$= a_0\ 10\ b_0 + a_1 (-5\ b_1 c_0 - 15\ b_0 c_1) + b_1\ 8\ a_2 + c_0\ 2\ a_0$$

El conjunto de valores de las variables continuas a_i, b_j, c_k que maximiza la función de consistencia global indica (cuanto más se aproximen a la unidad) cuál de las etiquetas correspondientes de cada nodo satisface mejor las relaciones prefijadas de compatibilidad. Se puede plantear así un problema de optimización con variables independientes continuas para resolver el problema planteado de compatibilidad en el diseño que es de naturaleza eminentemente discreta.

En este problema continuo el soporte global o consistencia global es la **función objetivo** que se desea maximizar; las **variables independientes** son las variables continuas asociadas a todas las etiquetas de todos los nodos; las **ligaduras** consisten en que la suma de las variables asociadas a las etiquetas de cada nodo tienen que sumar 1; las **restricciones** consisten en que el valor de cada variable asociada a cada etiqueta tiene que estar entre 0 y 1.

El problema así planteado presenta ciertas ventajas y ciertos inconvenientes. Entre las ventajas se pueden citar:

- Se trata de un problema resoluble por búsquedas locales.

- Las expresiones de las derivadas parciales son sencillas, y por tanto también el gradiente.

- Se pueden añadir relaciones nuevas con facilidad, e incluso etiquetas nuevas.

Entre los inconvenientes principales se pueden citar varios que hacen que la solución del problema sea lenta y muy inestable:

- El número de variables independientes es muy alto.

- La influencia de la formulación de las relaciones de compatibilidad es difícil de estimar.

- La función objetivo presenta muchos óptimos locales.

- El óptimo local alcanzado depende mucho del punto inicial y de la estrategia de búsqueda seguida; por ejemplo, el orden en que se van alterando los valores de las etiquetas de cada nodo influye mucho.

5.3.3 - Búsqueda en Espacio Heurístico.

La idea consiste en trasladar un problema de variables independientes discretas a otro espacio de variables independientes con propiedades de variables continuas, que es el espacio heurístico. La diferencia con el etiquetado consistente estriba en que se reduce el número de variables en vez de aumentarlo. Su limitación consiste en que sólo se podrá aplicar a problemas que cumplan unas condiciones especiales.

Para que este enfoque sea eficaz es necesario aplicar criterios basados en el conocimiento relativo a cada problema particular, pues es necesario jerarquizar y agrupar las variables independientes según su importancia relativa y según las restricciones. Además hay que resolver el problema de cómo almacenar y recuperar los puntos ya evaluados para evitar evaluar

un punto más de una vez. Las búsquedas en árbol permiten evitar esto mediante su forma natural de organización, pero no así las búsquedas locales.

Sea un vector de m variables independientes discretas, en el que el tamaño de cada variable x_i (número de valores posibles) vale n_i:

$$X = \{ x_1, x_2, ..., x_m \}$$

Supóngase que las variables independientes se pueden agrupar según su naturaleza en G grupos disjuntos, tales que cada grupo Z_i tenga g_i variables:

$$X = \{ Z_1, Z_2, ..., Z_G \}$$

tal que

$$Z_1 = \{ x_1, x_2, ..., x_{g1} \}$$
$$Z_2 = \{ x_{g1+1}, ..., x_{g1+g2} \}$$
$$\ldots \quad \ldots \quad \ldots$$
$$Z_G = \{..., x_{m-1}, x_m \}$$

Los criterios para agrupar las variables y permitir la definición del correspondiente espacio heurístico están encaminados a conseguir un buen **comportamiento local** de las variables independientes del espacio heurístico, y son:

- El valor de todo grupo Z_i tendrá una importancia relativa comparable a la de cada uno de los demás grupos con respecto a las restricciones y los atributos del problema.

- Las variables que pertenecen al mismo grupo admiten una jerarquía interna, de tal manera que los cambios en el valor de las variables de jerarquía superior influyan más en restricciones y atributos que los cambios en las variables de jerarquía inferior.

Una vez agrupadas las variables de esta forma, a cada grupo de variables se le asigna un eje del espacio heurístico. Cada punto de dicho eje corresponderá a una combinación particular de valores de las variables del grupo. Por lo tanto para un grupo Z de variables el número N total de puntos de su eje correspondiente (y por tanto el **tamaño** de la nueva variable independiente asociada) será:

$$N = \prod_{i=1}^{g} n_i$$

siendo

g	número de variables del grupo
n_i	tamaño de la variable i (número de valores distintos)

Las nuevas variables del espacio heurístico son tantas como grupos de variables discretas tenga el problema original. Son también variables discretas, pero tienen un tamaño mayor que las originales y, si se cumplen los dos criterios de agrupamiento citados, tendrán un buen comportamiento local, por lo que se podrán aplicar métodos de búsqueda local para la resolución del problema.

Por ejemplo, en el caso propuesto en el apartado anterior se puede suponer que las tres variables { A, B, C } forman un grupo de variables, ordenadas así según sus importancias relativas. El tamaño de la variable asociada en el espacio heurístico será:

$$N = n_a \; n_b \; n_c = 3 \times 2 \times 2 = 12$$

Los puntos del eje correspondiente a dicha variable en el espacio heurístico serán:

0	...	{ a_0, b_0, c_0 }
1	...	{ a_0, b_0, c_1 }
2	...	{ a_0, b_1, c_0 }
...	...	... ...
10	...	{ a_2, b_1, c_0 }
11	...	{ a_2, b_1, c_1 }

Si la jerarquía de variables es correcta, la distancia en el eje corresponde a una "distancia heurística", es decir, los puntos próximos presentan características similares. Esto permite interpolar y extrapolar, y por tanto llevar a cabo búsquedas locales eficaces (especialmente si en todos los ejes se toman incrementos igualmente significativos).

El problema de la codificación de puntos para evitar evaluaciones repetidas se resuelve del mismo modo: el valor de cada grupo de variables se puede representar por el valor de su variable heurística asociada, es decir, por un único número entero.

Para codificar y decodificar hace falta conocer el tamaño de cada variable original; por ejemplo, los tamaños de las variables del caso utilizado en este apartado son respectivamente 3, 2 y 2, y la codificación de un punto { a_i, b_j, c_k } resulta:

$$\{ a_i, b_j, c_k \} \quad \ldots \quad k + (2)\, j + (2)\,(2)\, i$$

La decodificación se hará naturalmente dividiendo sucesivamente por 2 y por 4 y apartando los restos de las divisiones: se trata simplemente de un problema de cambio de base. Estos problemas se llaman "transformaciones de claves" en [Wirth, 80].

La transformación del problema a un espacio heurístico resulta eficaz si realmente es posible organizar las variables según los criterios mencionados, es decir, si se consigue que las nuevas variables heurísticas presenten un buen comportamiento local. Si esto es así, se podrá reducir espectacularmente el número de variables, además de permitir la resolución de problemas de optimización con variables discretas mediante búsquedas locales.

Aparte de esto, el sistema de codificación y decodificación puede resultar muy útil para almacenar soluciones previas e implantar sistemas de memorias selectivas, como los mencionados en el Capítulo 7.

5.4 - Notas sobre Aplicaciones a Proyectos.

Las ideas y metodologías expuestas en este capítulo están inspiradas en el trabajo llevado a cabo en proyectos reales de investigación y desarrollo. En este apartado se comenta el trabajo realizado en dichos proyectos a la luz de las ideas y la nomenclatura utilizadas aquí, como si éstos fueran consecuencia de la tesis y no al contrario. De esta forma las aplicaciones reales proporcionan un respaldo experimental a las ideas y metodologías expuestas en la tesis. Con este objeto el orden de las notas no es cronológico, sino que sigue el de la exposición del capítulo.

En [Cuadra et al, 85] y [García et al, 87] se estableció una jerarquía de tres niveles para las variables independientes. Los criterios empleados en la jerarquización fueron la necesidad de llevar a cabo todas las búsquedas en espacios de dimensión constante y el "tamaño" de las variables independientes (muy discretas, discretas y continuas).

En la primera fase de diseño, que utilizaba el modelo más simplificado del sistema, una de las optimizaciones parciales se resolvía de forma no iterativa. Por medio de un conjunto de

reglas (if...then...), y a partir de los valores de los requisitos de diseño, se ofrecía analíticamente el punto óptimo. Estas reglas y las expresiones analíticas se obtuvieron mediante un estudio previo de la monotonía de las funciones y restricciones que intervenían en el modelo más simplificado del sistema.

El comportamiento local de las variables era bueno en todos los niveles. En cuanto al tamaño de las variables (número de valores distintos en la práctica), en el primer nivel era 5, en el segundo era 15 y en el tercero oscilaba entre 200 y 1000, según las distintas variables del espacio. Dado el buen comportamiento local de las variables se utilizó el algoritmo de Hooke & Jeeves en todos los espacios de búsqueda, aunque con ligeras modificaciones en los casos de variables discretas.

El usuario podía elegir el tamaño de las variables continuas mediante los incrementos mínimos (precisión) deseados para cada una de ellas, aunque por supuesto se mantenían unos valores por omisión almacenados en ficheros. También se podía fijar el valor de cualquier variable independiente, eliminando grados de libertad del problema.

En [IIT, 87] se realizó un estudio de viabilidad de carácter general y un prototipo como demostración de las técnicas estudiadas. En el prototipo se implantó una sola fase de diseño descompuesta en dos optimizaciones parciales anidadas; la de nivel superior correspondía a variables de tamaño pequeño (entre 2 y 10) y de mal comportamiento local, mientras que la inferior se llevaba a cabo en un espacio de variables de tamaño grande (entre 100 y 1000) y de buen comportamiento local.

En la optimización parcial de nivel superior se planteó una búsqueda en árbol de soluciones completas guiada por reglas de sensibilidad. En la de nivel inferior se empleó el algoritmo de Hooke & Jeeves para llevar a cabo búsquedas locales con respecto a distintas funciones objetivo (por ser un problema multiatributo).

En el desarrollo de la herramienta creadora de ábacos lineales (CAL) se empleó el algoritmo de Hooke & Jeeves para encontrar los puntos de máximo error entre la función real y el ábaco provisional; los nuevos "puntos muestra" del ábaco se colocan precisamente donde se produce el máximo error provisional.

En [Cuadra y León, 90] se empleó una búsqueda en árbol de soluciones incompletas; la selección de nodos era multiatributo, con un atributo real (volumen ya aprovechado) y un atributo heurístico (volumen aprovechable) para evaluar el futuro de las distintas ramas. En la

expansión de un nodo, para cada nodo hijo se fijaba un valor distinto de tres variables independientes jerarquizadas.

Capítulo 6

Optimización Multiatributo

Ya se ha expuesto en otros capítulos de la tesis que el resultado de una optimización multiatributo no es un sólo punto, sino un **conjunto de puntos** de interés que en el espacio de atributos forman parte de la **hipersuperficie óptima**. Estos puntos pueden además pertenecer a distintas regiones de búsqueda del espacio de variables independientes.

También se ha comentado ya que una optimización multiatributo en general requiere exploraciones en distintas regiones de búsqueda, a partir de distintos puntos iniciales y con distintos criterios de optimización, todo ello para encontrar un conjunto de puntos de interés. Dicho conjunto puede filtrarse con criterios de **selección multiatributo** para reducir el número de puntos sin perder los más importantes.

En este capítulo se tratan con detalle todas estas ideas siguiendo el esquema que muestra la figura 6.1. La optimización multiatributo consta, como tal optimización, de los procesos de evaluación y búsqueda. La evaluación multiatributo se trata en el apartado 6.1, mientras que la búsqueda multiatributo se expone en el apartado 6.3. Según el tipo de búsqueda elegida, local o en árbol, el enfoque es distinto: en la búsqueda local se sustituye la búsqueda multiatributo por un conjunto de búsquedas monoatributo, que ya se han tratado ampliamente en el Capítulo 5 (junto a las búsquedas en árbol). En ambos tipos de búsqueda juega un papel fundamental la selección multiatributo, tratada con detalle en el apartado 6.2.

En el apartado 6.4 se expone brevemente cómo puede llevarse a cabo una optimización multiatributo en un sistema complejo descompuesto en optimizaciones parciales, tanto anidadas como paralelas.

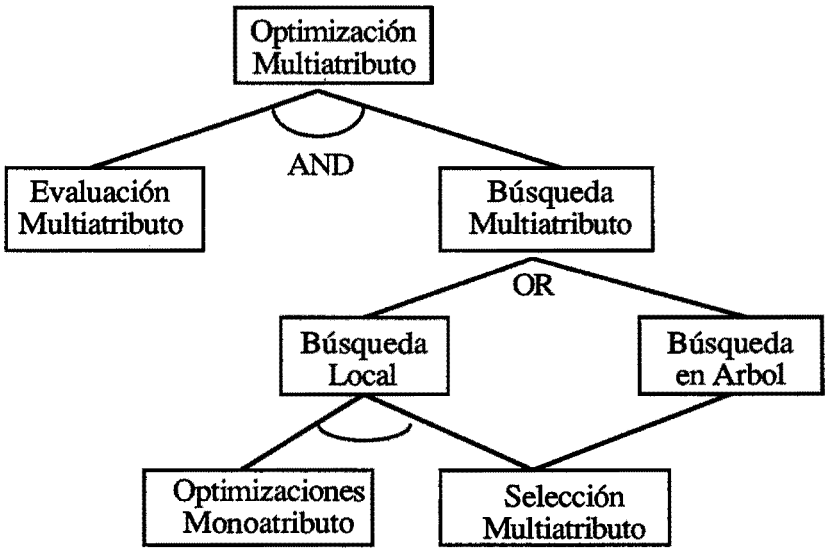


Figura 6.1. Enfoque propuesto para la Optimización Multiatributo

Finalmente, el apartado 2.6 está dedicado a comentar aspectos relacionados con este capítulo de los proyectos reales de diseño por ordenador que han dado lugar a esta tesis

6.1 - Evaluación Multiatributo: Hipersuperficie Optima.

La **evaluación multiatributo** se ha descrito ya en el Capítulo 3. Consiste simplemente en asociar a cada punto X de la región de búsqueda un punto E(X) del espacio de estado de las restricciones y un punto A(X) del espacio de atributos, según muestra la figura 6.2. Obsérvese que puntos muy distantes en el espacio de variables independientes (o sea, soluciones muy diferentes) pueden estar muy próximos en el espacio de atributos o en el de restricciones.

El **tratamiento de las restricciones** en la búsqueda admite distintas soluciones. Una de ellas es descomponer la búsqueda en dos etapas: la primera, encargada fundamentalmente de alcanzar un punto de la región factible, dando una importancia relativamente pequeña a los atributos frente a las restricciones violadas; la segunda, partiendo del punto factible hallado en la primera etapa, buscaría puntos óptimos sin salirse de la región factible.

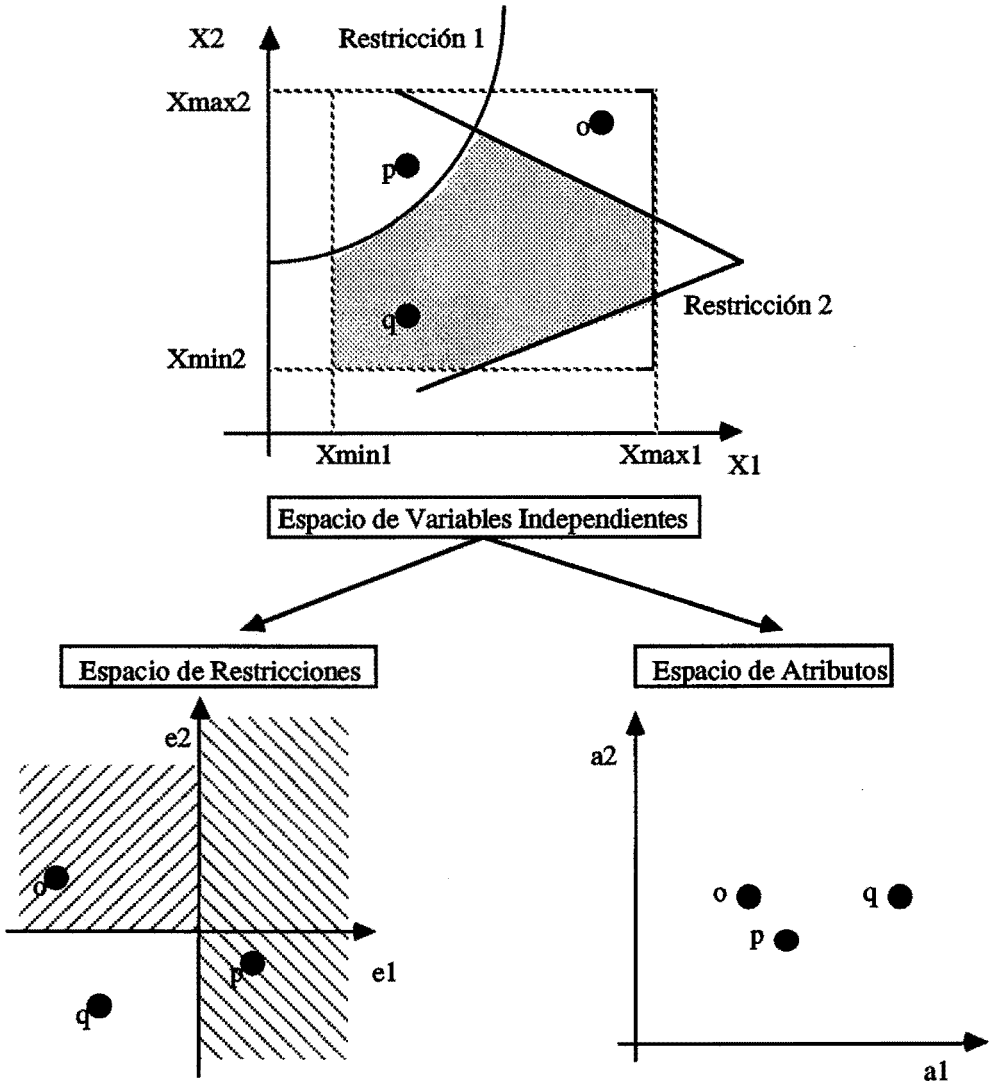


Figura 6.2. Evaluación Multiatributo.

Otra solución consiste en agrupar el estado de las restricciones en uno o varios atributos ficticios que se desea minimizar, suponiendo que tienen un valor positivo en caso de restricciones violadas y cero en la región factible. A estos atributos ficticios se les puede conceder una importancia relativa variable para guiar la búsqueda suavemente hacia puntos factibles pero a la vez cercanos al óptimo.

Nota bibliográfica:
El tratamiento de las restricciones como atributos ficticios (penalizaciones) es una idea ya clásica; por ejemplo, en [Gill et al, 81] se describen distintas formas de definir las penalizaciones (funciones lineales, cuadráticas, logarítmicas, etc.)

El tratamiento de las restricciones como atributos ficticios (penalizaciones) es la base de la técnica llamada constraint relaxation, aplicada al diseño en [Navinchandra & Marks, 87]. Los atributos ficticios asociados a las restricciones se llaman deviation variables en [Kamal et al, 87].

En cualquier caso el problema se puede reducir siempre a movimientos a un espacio (quizá ampliado) de atributos, y este espacio es el que se va a estudiar en este capítulo. En este espacio, es fundamental el concepto de hipersuperficie óptima. Se propone aquí una definición de **hipersuperficie óptima**, basada en la intuición:

> "Un punto X pertenece a la hipersuperficie óptima del espacio de atributos A cuando todos los puntos de un determinado entorno de A(X) que presentan un valor mejor en al menos uno de los atributos presentan un valor peor en al menos otro de los atributos."

Es importante señalar que la hipersuperficie óptima no será en general una auténtica hipersuperficie desde un punto de vista analítico o geométrico. De hecho suele estar constituida por un conjunto disperso de puntos, debido por ejemplo a que las variables pueden ser discretas o a la influencia de ligaduras y restricciones. Sin embargo se usa aquí el término de hipersuperficie por representar intuitivamente la frontera de un conjunto de puntos.

La definición de hipersuperficie óptima propuesta permite que la forma de la hipersuperficie óptima dependa del **tamaño** del entorno y de los conceptos de **mejor** y **peor**, dejando que las características de cada aplicación sugieran la elección más adecuada. A modo de ejemplo, la figura 6.3 muestra una posible hipersuperficie óptima en un caso en que se desea minimizar dos atributos y otra en un caso en que se desea maximizar dos atributos.

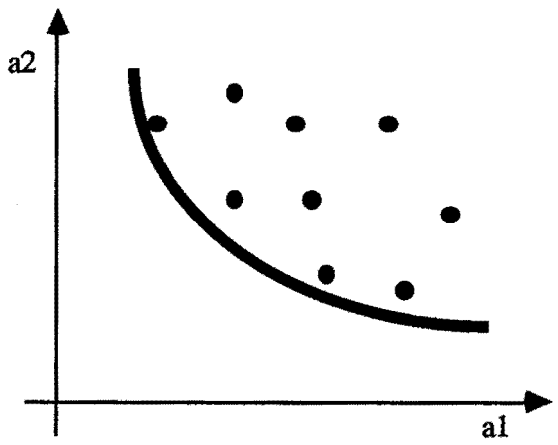


Figura 6.3.a. Hipersuperficie Optima minimizando dos atributos.

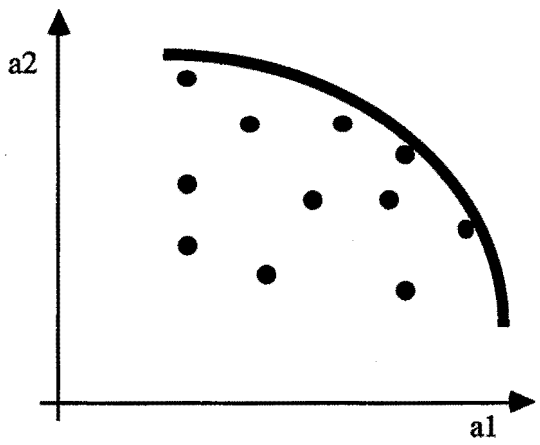


Figura 6.3.b. Hipersuperficie Optima maximizando dos atributos.

Nota bibliográfica:
La hipersuperficie óptima está estrechamente ligada al concepto económico de "allocative efficiency" introducido por Vilfredo Pareto en 1906. En [Samuelson & Nordhaus, 85] se ofrece la siguiente definición:

> "Allocative efficiency occurs when there is no possible organization of production that would make everyone better off... Under conditions of efficiency, therefore, one person's utility can be increased only by lowering someone else's utility."

Los atributos son aquí la satisfacción económica de distintas personas, industrias o sectores económicos. Los puntos que cumplen la condición de "allocative efficiency" se llaman también "Pareto-efficient" y están situados sobre la llamada "utility-possibility frontier, equivalente a la hipersuperficie óptima.
La hipersuperficie óptima recibe distintos nombres en la literatura técnica: non inferior set en [Chankong & Haimes, 83]; knee set en [Schweppe & Merril, 86]; non dominated set of alternatives en [Navinchandra & Marks, 87]; compromise solution set en [Jazdzynski, 89]; trade-off curve (or surface) en [Kermanshahi et al, 90]; non inferior solution set en [Kermanshahi et al, 90] y trade-off curve set en [Schweppe & Merril, 86].

Una hipersuperficie óptima puede constar teóricamente de infinitos puntos, pero en la práctica el problema de hallarla se reduce a seleccionar un conjunto reducido de puntos representativos de la hipersuperficie de entre un número finito de candidatos. El criterio de selección seguido (tamaño del entorno, conceptos de mejor y peor) definirá la forma de la hipersuperficie. En el apartado siguiente se describen tres criterios alternativos de selección multiatributo.

6.2 - Selección Multiatributo: Distintas Hipersuperficies Optimas.

Las dos primeras definiciones de hipersuperficie óptima (dominación estricta y dominación significativa) proceden de un intento puro de selección de entre un gran número de soluciones ya generadas, y se han aplicado en planificación de la generación eléctrica a largo plazo [Schweppe & Merril, 86].

La tercera definición que se presenta (optimalidad respecto a una familia de funciones objetivo) deriva del proceso de optimización multiatributo seguido en [IIT, 87], y como se verá está orientada a generar directamente los puntos de la hipersuperficie óptima en vez de seleccionar entre soluciones ya existentes.

6.2.1 - Dominación Estricta y Dominación Significativa.

Los conceptos de dominación estricta y dominación significativa están tomados del procedimiento llamado **Multiple Attribute Trade-off Analysis** [Schweppe & Merril, 86],

aplicado a la planificación de inversiones a largo plazo del sistema de generación de energía eléctrica.

La idea consiste en lo siguiente: se seleccionan manual o automáticamente un número elevado (hasta de varios miles) de conjuntos completos de variables independientes de definen otros tantos planes alternativos de inversiones. Todos los planes se evalúan con respecto a un conjunto de atributos que normalmente son mutuamente conflictivos, y por tanto no se pueden reducir todos a una sola función objetivo: coste, fiabilidad (o más concretamente probabilidad de energía no suministrada), impacto ambiental, etc. Una vez situados todos los planes en el espacio de atributos, se procede a realizar una selección multiatributo que reduce el número de los mismos a una magnitud razonable. De entre los planes supervivientes, un grupo de expertos seleccionará el más interesante según sus propios criterios.

La figura 6.4 muestra gráficamente los dos criterios alternativos de selección mencionados. Supuesto un criterio de dominación y dado un plan concreto, todos aquellos planes "dominados" por él serán rechazados. La hipersuperficie óptima estará compuesta por aquellos planes no dominados por ningún otro plan. La figura 6.6 muestra cómo la hipersuperficie óptima puede variar según el criterio de dominación elegido.

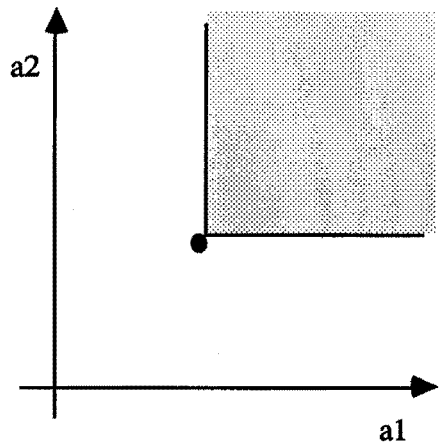


Figura 6.4.a. Región dominada por una solución según criterio de dominación estricta.

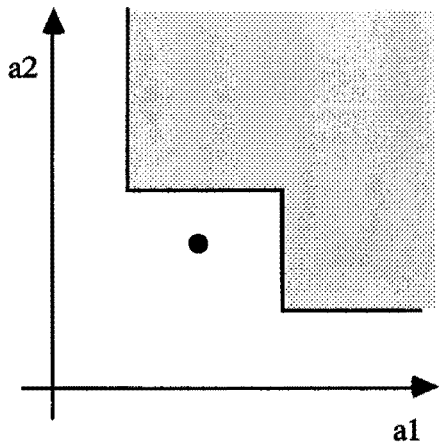


Figura 6.4.b. Región dominada por una solución según criterio de dominación significativa.

El criterio de dominación **significativa** (figura 6.4.b) es más refinado que el de dominación **estricta** (figura 6.4.a) en dos aspectos: primero, rechaza puntos en los que la mejora en uno de los atributos no compensa lo que empeoran los otros; segundo, no rechaza

puntos muy próximos en el espacio de atributos, aunque estén dominados según el criterio estricto. Con respecto a este último aspecto conviene destacar que los puntos que están muy próximos tanto en el espacio de variables independientes como en el de atributos sí deberían ser rechazados, pues en realidad representan soluciones muy parecidas entre sí.

Desde un punto de vista de optimización, el proceso planteado en [Schweppe & Merril, 86] no es muy apropiado, pues podrían buscarse métodos más eficaces de generar los planes candidatos. Sin embargo, establece unos principios generales muy útiles para la selección multiatributo y por tanto para la optimización multiatributo. Por ejemplo, la selección de soluciones por dominación estricta se aplicó con éxito a la optimización de la distribución de objetos en un espacio limitado en [Cuadra & León, 90].

6.2.2 - Optimalidad Respecto a una Familia de Funciones Objetivo.

Como ya se indicó en el Capítulo 3, una **función objetivo** es una función escalar de variable vectorial cuyo valor se desea optimizar, y cuyos argumentos son el vector de atributos A(X) y el vector de estado de las restricciones E(X). Dada la posible expresión del estado de las restricciones como atributos ficticios, se puede suponer que una función objetivo depende sólo del vector de atributos:

$$f = f(A(X)) = f(a_1(X), a_2(X), ..., a_n(X)) = f(a_1, a_2, ..., a_n)$$

Una función objetivo representa un criterio de optimización particular, sirviendo para llevar a cabo una optimización monoatributo en la cual el valor de la función hace el papel de atributo. Ahora bien, si a cada atributo a_i se le asocia un parámetro λ_i cuyo valor exprese la importancia relativa de dicho atributo frente a los demás, se puede definir una familia de funciones objetivo que represente una familia de criterios de optimización:

$$F(a_1, a_2, ..., a_n \mid \lambda_1, \lambda_2, ..., \lambda_n)$$

Una familia de funciones objetivo en teoría consta de infinitas funciones, aunque en la práctica se utilizará sólo un conjunto reducido de funciones representativas de dicha familia.

Una vez establecida una familia de funciones objetivo se puede definir la **hipersuperficie óptima** como aquel conjunto de puntos que son óptimos globales al menos con respecto a una función objetivo de dicha familia.

La familia de funciones objetivo más inmediata e intuitiva es la de las posibles combinaciones lineales de los atributos, empleada en optimización de diseño en [IIT, 87]:

$$F = F(a_1, a_2, ..., a_n \mid \lambda_1, \lambda_2, ..., \lambda_n) = \lambda_1 a_1 + \lambda_2 a_2 + ... + \lambda_n a_n$$

La hipersuperficie óptima quedaría así aproximada por hiperplanos tangentes. La figura 6.5 muestra una selección multiatributo de puntos según el criterio de optimalidad respecto a una familia de funciones objetivo lineales.

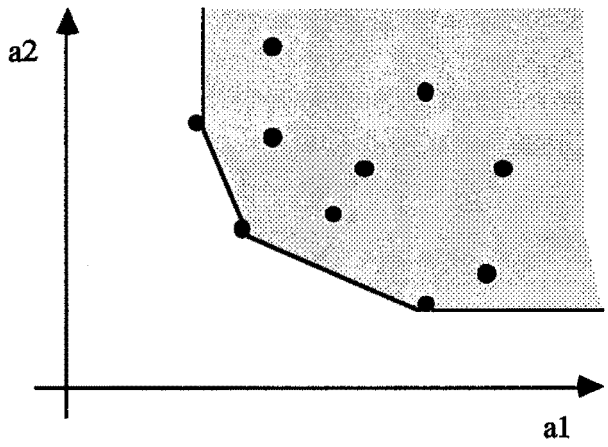


Figura 6.5. Selección Multiatributo según criterio de optimalidad respecto a una familia de funciones lineales.

Finalmente, la figura 6.6 muestra cómo el conjunto de puntos representantes de la hipersuperficie óptima puede ser distinto según la definición de la misma que se emplee, es decir, según el criterio de selección multiatributo que se escoja.

6.2.3 - La Incertidumbre en la Selección Multiatributo.

Cuando se abordan problemas de planificación la incertidumbre en la evaluación de soluciones es un factor crítico. Por ejemplo, en [Schweppe & Merril, 86] sólo se conocen estimaciones de la pluviosidad, la evolución de la demanda, los costes de combustibles, etc. En el campo del diseño la incertidumbre también existe, aunque de forma más limitada. Por

ejemplo, los costes de los materiales pueden variar durante el horizonte previsto de fabricación de una pieza.

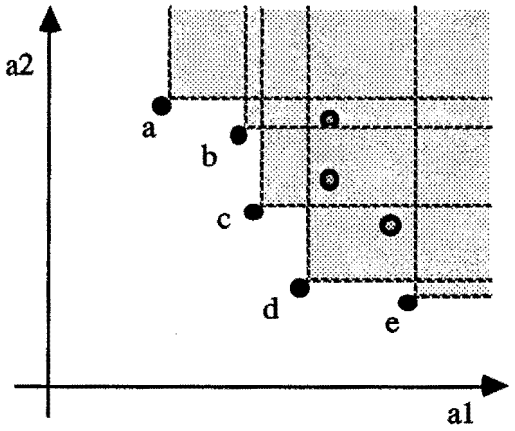


Figura 6.6.a. Dominación Estricta.

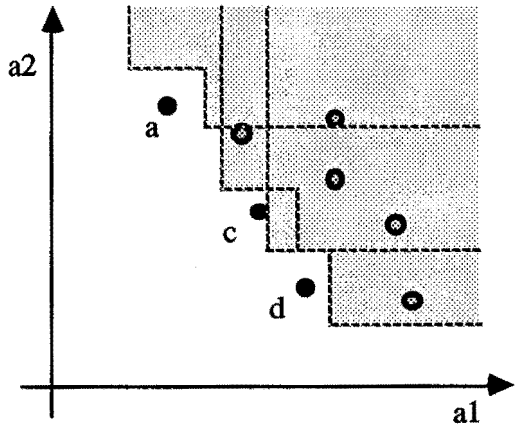


Figura 6.6.b. Dominación Significativa.

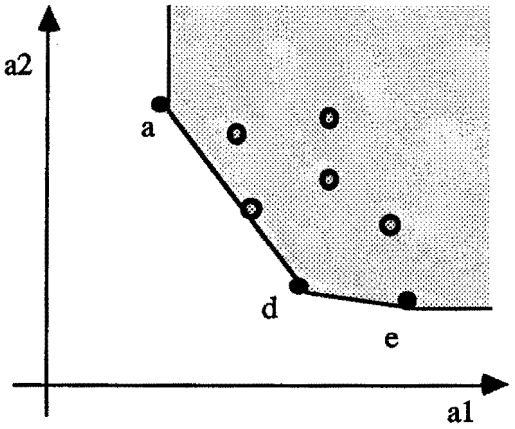


Figura 6.6.c. Optimalidad respecto a una familia de funciones objetivo lineales.

Figura 6.6. Distintas hipersuperficies óptimas según distintos criterios de selección multiatributo.

Un caso de incertidumbre en la evaluación se da cuando el proceso de optimización seguido se basa en una búsqueda en árbol de soluciones incompletas; los heurísticos empleados para evaluar los nodos serán normalmente imprecisos, por lo que la selección multiatributo debe considerar un cierto nivel de incertidumbre.

La fuente de incertidumbre más importante en diseño es, no obstante, la falta de precisión de los modelos utilizados en cada fase de diseño. Esto hace que la posición de cada solución en el espacio de atributos no sea exacta, sino que esté sujeta a errores a veces importantes.

Dado que en todas las fases de diseño habrá que realizar selecciones multiatributo, los criterios de selección deben tener en cuenta los errores de modelado y ser conservadores al rechazar soluciones próximas a la hipersuperficie óptima.

Una solución posible es definir márgenes de incertidumbre que suavicen los criterios de selección. Estos márgenes serán más amplios cuanto menos preciso sea el modelo empleado en la evaluación de soluciones, y serán distintos para cada atributo según los errores estimados para cada uno de ellos.

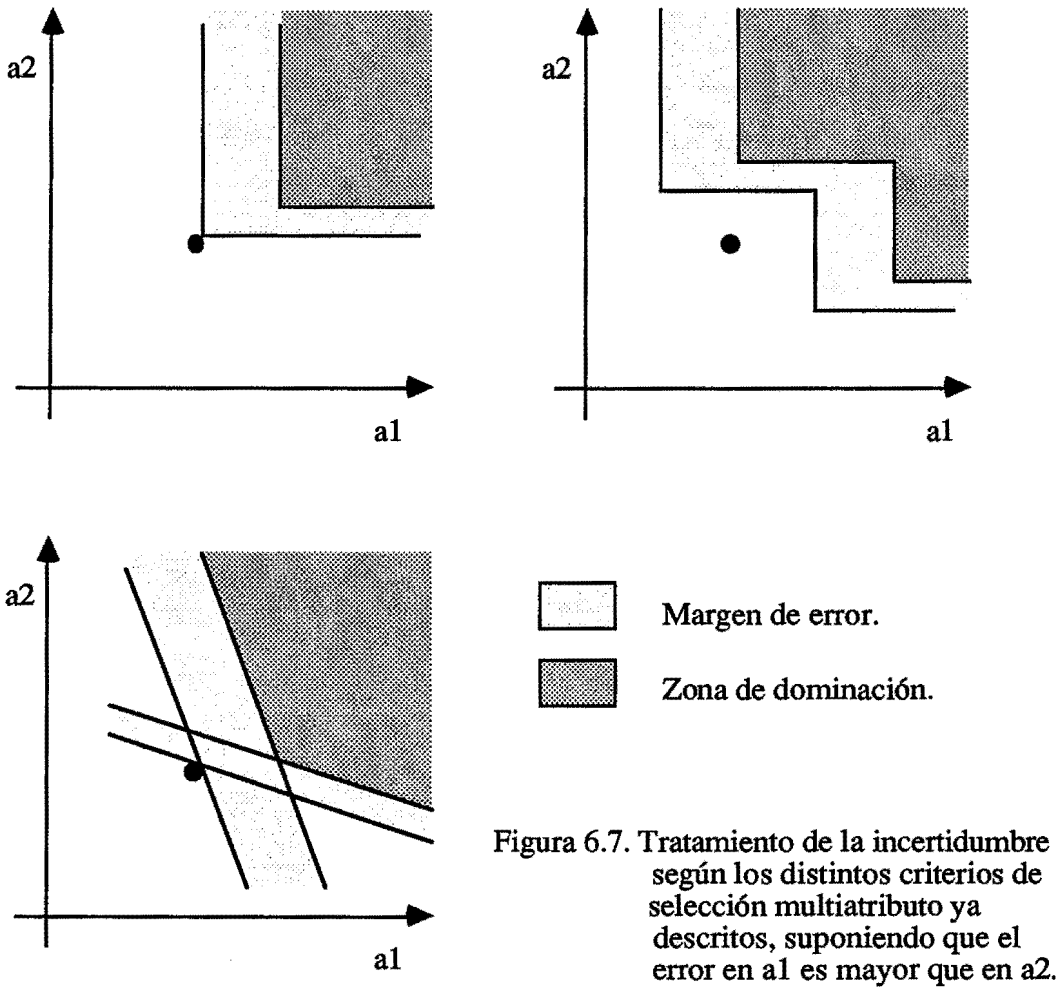


Figura 6.7. Tratamiento de la incertidumbre según los distintos criterios de selección multiatributo ya descritos, suponiendo que el error en a1 es mayor que en a2.

Nota bibliográfica:
El diseño de circuitos integrados es un campo en el que la incertidumbre por falta de precisión en el modelado es un grave problema. En las primeras fases del diseño se tienen ya que tomar decisiones en función de las características finales del producto (atributos y restricciones) como por ejemplo el espacio que ocupa el circuito o el tiempo máximo de respuesta; sin embargo, esto no se puede saber con precisión hasta que se ha resuelto el problema del routing con gran detalle, es decir, hasta las últimas fases del diseño. Este problema se conoce como multilevel specifications o interlevel constraints en trabajos sobre diseño de VLSI, como [McFarland et al, 90].
En [Schweppe & Merril, 86] se proponen dos tratamientos alternativos para la incertidumbre:
- El primero, basado en medidas de probabilidad, consiste en definir la relación de dominación con respecto a una probabilidad dada P; por ejemplo, el plan A domina al B si la probabilidad de que A domine a B (según un criterio de dominación dado) es igual o mayor que P.
- El segundo, llamado "unknown but bounded", maneja los límites superior e inferior de cada incertidumbre, usando por tanto el margen de incertidumbre descrito aquí.

La figura 6.7 muestra ejemplos de los márgenes de error para los distintos criterios de selección que aquí se han descrito. En estos ejemplos se ha supuesto que el error estimado en el atributo a_1 es mayor que el estimado en el atributo a_2.

Nótese que los márgenes de error no deben calcularse según el error absoluto de cada atributo, sino según el error relativo de la diferencia entre atributos (esto ya se indica en [Schweppe & Merril, 86]). Un ejemplo: si la incertidumbre en el coste de fabricación de un motor se debe al precio del cobre, al comparar dos diseños alternativos dicha incertidumbre influirá si los dos diseños necesitan distinta cantidad de cobre. Si necesitaran la misma cantidad, el desplazamiento correspondiente a una variación de precio del cobre afectaría a ambos por igual en el espacio de atributos.

6.3 - Búsqueda Multiatributo.

Una optimización es un proceso sistemático de búsqueda y evaluación. En una optimización monoatributo se busca un punto óptimo que a ser posible sea un óptimo global, y no sólo local. En una optimización multiatributo la búsqueda pretende hallar un conjunto significativo de puntos de la hipersuperficie óptima. Tanto en una como en otra el objetivo se puede lograr por medio de búsquedas locales (cada una monoatributo) o búsquedas en árbol, ya que el tipo de búsqueda lo sugiere el tipo de variables independientes del problema.

Nota bibliográfica:
[Chankong & Haimes, 83] es un trabajo excelente que recoge y clasifica diversas técnicas matemáticas para hallar la hipersuperficie óptima (denominada non inferior set) e incluso para seleccionar un óptimo de compromiso. Dadas ciertas condiciones matemáticas (linealidad, convexidad) la hipersuperficie óptima puede ser definida analíticamente; desgraciadamente, los problemas reales de optimización de diseño no suelen cumplir dichas condiciones.

Un método citado en [Chankong & Haimes, 83] es el llamado Goal Programming, aplicado a la optimización de planes de inversión en [Nanda et al, 87] y [Kermanshahi et al, 90].

6.3.1 - Búsqueda Multiatributo Mediante Búsquedas Locales.

Una **búsqueda local** es aconsejable cuando las variables independientes tienen un **tamaño grande** y presentan un **buen comportamiento local**. Sólo es válida para resolver **optimizaciones monoatributo**, pues se basa en movimientos de avance en la dirección aparentemente más interesante (por ejemplo la de mayor gradiente).

Si se desea seguir varias direcciones de búsqueda simultáneamente, el proceso se convierte en una búsqueda en árbol, aunque en cada una de las ramificaciones de dicho árbol se sigan los principios de una búsqueda local.

Supóngase un conjunto de n_p puntos iniciales, pertenecientes en general a distintas regiones de búsqueda, a partir de los cuales se quiere realizar una búsqueda multiatributo. Si se define una familia de funciones objetivo:

$$F(a_1, a_2, ..., a_n \mid \lambda_1, \lambda_2, ..., \lambda_n)$$

de la cual se selecciona un conjunto de n_f funciones objetivo representativas, se puede hallar un máximo de N puntos distintos de la hipersuperficie óptima mediante N búsquedas locales, siendo N :

$$N = n_p n_f$$

Esto significa que desde cada punto inicial se tiene que llevar a cabo una **optimización monoatributo** respecto a cada una de las funciones objetivo, dentro de la región de búsqueda correspondiente a cada punto inicial concreto. Esta es la solución aplicada en [IIT, 87] a la optimización de diseño.

Tras encontrar todos los óptimos locales se puede intentar reducir el número de puntos eliminando aquellos que no pertenezcan a la hipersuperficie óptima mediante una **selección multiatributo**. Debe tenerse en cuenta que los óptimos locales no siempre son globales, debido a que el punto inicial o la estrategia de búsqueda local seguida no sean adecuados. Además de esto, el criterio de selección multiatributo podrá ser muy duro por necesidades de tiempo o memoria.

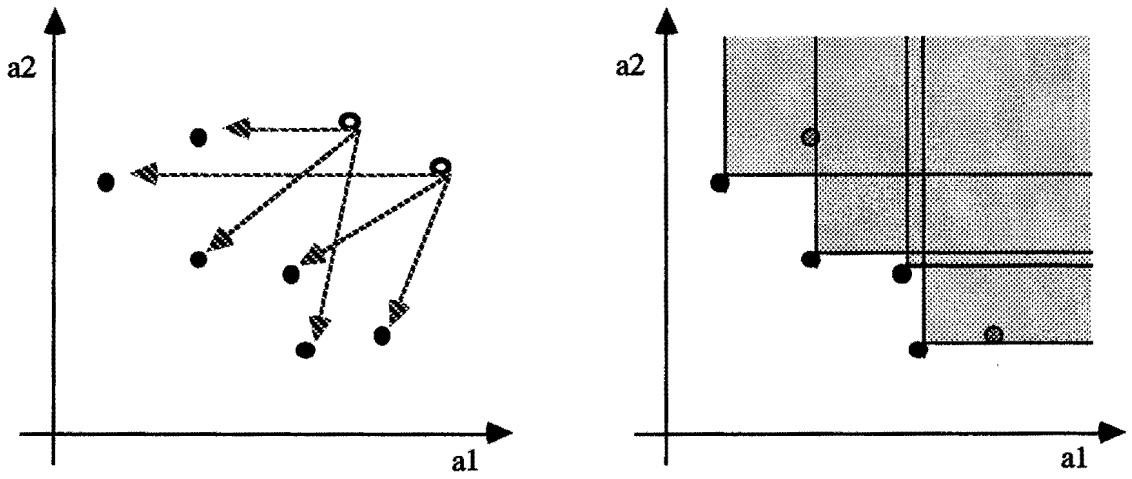


Figura 6.8. Conjunto de búsquedas locales monoatributo + selección multiatributo.

La figura 6.8 muestra un caso ejemplo de dos atributos, con dos puntos iniciales y tres funciones objetivo lineales. Esto implica seis optimizaciones monoatributo por búsqueda local que producen seis puntos candidatos a representar a la hipersuperficie óptima. La misma figura muestra una selección multiatributo posterior por dominación estricta que reduce a cuatro el número de puntos representativos.

Como se aprecia en la figura 6.8, un conjunto de búsquedas locales se puede interpretar como una búsqueda en árbol con un sólo nivel de profundidad, tantas ramificaciones como funciones objetivo y tantos nodos iniciales como puntos iniciales haya.

6.3.2 - Búsqueda Multiatributo en Arbol.

Las **búsquedas en árbol** son adecuadas cuando las variables independientes son de un **tamaño pequeño** y presentan un **mal comportamiento local**. Se adaptan perfectamente a una optimización multiatributo, pues permiten avanzar en varias direcciones simultáneamente.

En una búsqueda en árbol, los nodos hijo se evalúan para seleccionar los mejores, pues por cuestiones de tiempo y memoria se desea sólo **expandir** (generar nodos hijo) **los mejores nodos**. La selección de nodos se puede hacer tanto en **profundidad**, es decir, sólo

compiten entre sí los nodos hijos del mismo nodo padre, como en **anchura**, donde compiten entre sí todos los nodos del mismo nivel de profundidad.

La selección de nodos admite tanto una ordenación monoatributo como una selección multiatributo, según cualquier definición de hipersuperficie óptima. La figura 6.9 muestra una búsqueda en árbol representada en el espacio de dos atributos. En cada nivel de profundidad del árbol se genera una nueva hipersuperficie óptima, y sólo los nodos pertenecientes a dicha hipersuperficie son expandidos.

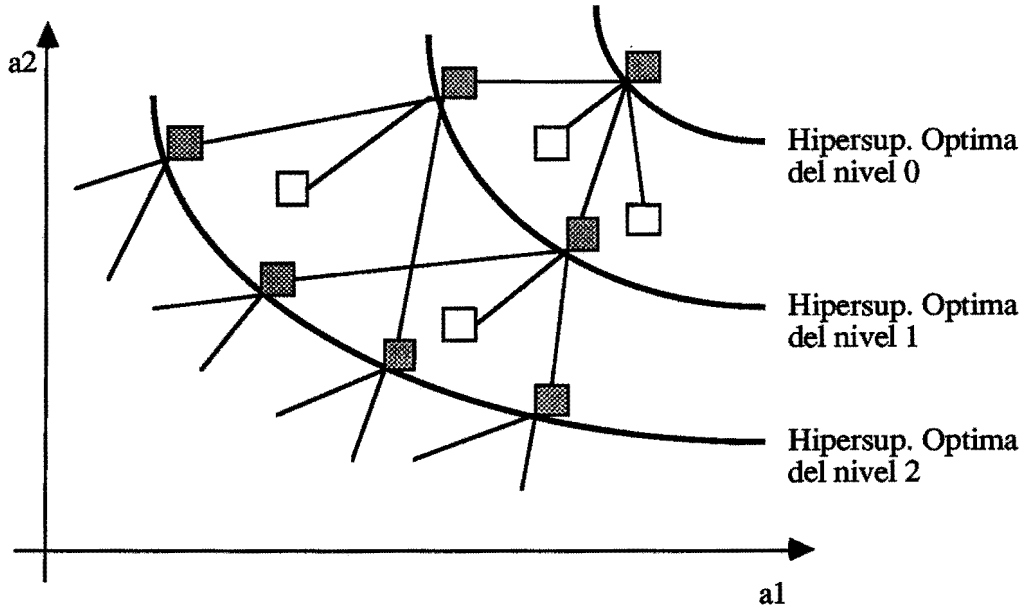


Figura 6.9. Búsqueda Multiatributo en Arbol: búsqueda en árbol con selección multiatributo.

En la figura 6.9 la selección multiatributo se hace en anchura, es decir, todos los nodos del mismo nivel de profundidad entran en competición entre sí, aunque sean hijos de distinto nodo padre. Las hipersuperficies óptimas de distintos niveles que muestra la figura 6.9 se denominan "pareto subsurfaces" en [Navinchandra & Marks, 87].

En el Capítulo 5 se planteó la distinción entre dos tipos de búsqueda en árbol: la de soluciones completas y la de soluciones incompletas. En los árboles de soluciones incompletas los nodos se pueden seleccionar utilizando no sólo los atributos reales, sino también atributos ficticios (heurísticos) para compensar el hecho de no conocer todavía el valor de todas las

variables independientes. Es también aconsejable realizar una selección "blanda" como la aplicada en el tratamiento de la incertidumbre (apartado 6.2.3).

6.4 - Optimización Multiatributo y Optimizaciones Parciales.

En el Capítulo 4 se expone cómo una optimización compleja se puede descomponer en un sistema de **optimizaciones parciales** más sencillas, definiéndose las optimizaciones parciales paralelas y las anidadas. En este apartado se discute la compatibilidad de un sistema así con la optimización multiatributo.

Cuando se desea realizar una optimización multiatributo, lo más aconsejable es limitar la optimización multiatributo a la **optimización parcial principal** o de más alto nivel, llevándose a cabo optimizaciones monoatributo (una función objetivo cada vez) en las optimizaciones parciales de nivel inferior.

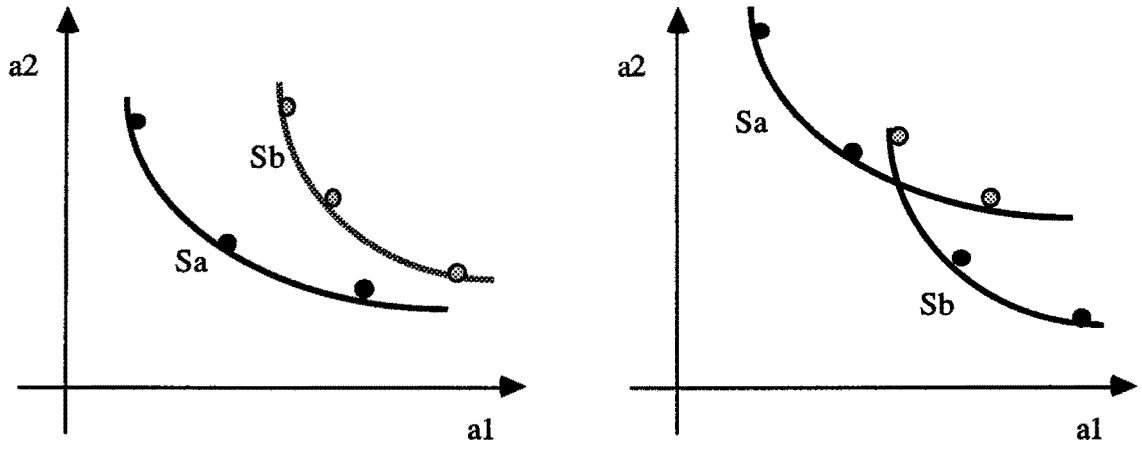
Por ejemplo, en el caso de optimizaciones parciales **paralelas** no se debe optimizar con respecto a varias funciones objetivo simultáneamente, pues se superponen los problemas de convergencia propios de las optimizaciones paralelas a la ramificación propia de la búsqueda multiatributo, produciendo una explosión combinatoria inadmisible.

En el caso de optimizaciones parciales **anidadas** pueden sin embargo llevarse a cabo optimizaciones multiatributo a distintos niveles. Esto implica que al evaluar un punto del espacio de variables independientes de una optimización de nivel superior se obtiene no un punto en el espacio de atributos, sino una hipersuperficie óptima parcial (representada por un conjunto de puntos), procedente de optimizaciones multiatributo de nivel inferior.

Por ejemplo, supóngase una variable de nivel superior X que puede tomar dos valores distintos, X_a y X_b. Para evaluar el valor X_a se lleva a cabo una optimización multiatributo en niveles inferiores, que produce como resultado la hipersuperficie óptima S_a. Del mismo modo, al evaluar el valor X_b se produce la hipersuperficie óptima S_b.

Para determinar cuál de los dos valores de X es mejor hay que comparar ambas hipersuperficies, naturalmente con criterios multiatributo. La figura 6.10 muestra como un valor de X puede ser rechazado porque todos los puntos de su hipersuperficie asociada sean

"dominados" por algún punto de la otra hipersuperficie, y cómo puede ocurrir que ambos valores deban conservarse por tener los dos al menos un punto representante de la hipersuperficie óptima conjunta.



Sa domina a Sb en conjunto, por lo que Xa es mejor que Xb.

Ninguna hipersuperficie domina a la otra, por lo que tanto Xa como Xb deben conservarse.

Figura 6.10. La optimización multiatributo en optimizaciones anidadas exige comparar hipersuperficies óptimas entre sí.

6.5 - Notas sobre Aplicaciones a Proyectos.

Las ideas y metodologías expuestas en este capítulo están inspiradas en el trabajo llevado a cabo en proyectos reales de investigación y desarrollo. En este apartado se comenta el trabajo realizado en dichos proyectos a la luz de las ideas y la nomenclatura utilizadas aquí. De esta forma las aplicaciones reales proporcionan un respaldo experimental a las ideas y metodologías expuestas en la tesis. Con este objeto el orden de las notas no es cronológico, sino que sigue el de la exposición del capítulo.

En [Cuadra et al, 85] y [García et al, 87] se llevaron a cabo optimizaciones multiatributo, pues se intentaba sólo minimizar el coste de fabricación. La violación de restricciones se reflejaba sumando penalizaciones (costes ficticios) al coste real, por lo que la optimización se

trataba en realidad como no restringida (unconstrained). Cada penalización era directamente proporcional a una estimación de tipo porcentual de la violación de la restricción correspondiente.

El usuario podía seleccionar entre 13 restricciones posibles cuáles se aplicaban al problema de , aparte naturalmente de las limitaciones en los rangos de las variables independientes. También podía fijar los distintos costes ficticios asociados a la violación de cada restricción.

En [IIT, 87] se llevó a cabo una optimización multiatributo con dos atributos mutuamente conflictivos: el coste total de fabricación y lanzamiento de un satélite y la fiabilidad del proyecto (probabilidad de que el satélite completara su tiempo de vida en pleno funcionamiento). La optimización multiatributo se realizó por medio de búsquedas locales con distintas funciones objetivo, que eran combinaciones lineales de los dos atributos (ver figura 6.8). De este modo la hipersuperficie (en este caso curva) óptima se aproximó por medio de rectas tangentes.

En [Cuadra y León, 90] se empleó una búsqueda en árbol de soluciones incompletas; la selección de nodos era multiatributo, con un atributo real (volumen ya aprovechado) y un atributo heurístico (volumen aprovechable) para evaluar el futuro de las distintas ramas. En la expansión de un nodo, para cada nodo hijo se fijaba un valor distinto de tres variables independientes jerarquizadas.

Esta solución es totalmente original, pues lo habitual es sumar los heurísticos y los atributos reales para evaluar los nodos según una función objetivo única. Esto implica que el número de nodos hijo debe ser fijado a priori o con un criterio de corte monoatributo; sin embargo, en la selección multiatributo (ver figura 6.9).el número de nodos seleccionados depende de las condiciones de dominación y por tanto de las características de cada nodo padre.

Capítulo 7

Sustitución de Funciones Lentas por Abacos

El enfoque que se presenta en esta tesis para abordar la optimización de diseño por ordenador está basado en procesos iterativos con distintas regiones de búsqueda, varios puntos iniciales, varias estrategias y funciones objetivo. Todo el procedimiento se apoya en la evaluación de un gran número de puntos.

Aunque se descomponga el proceso de optimización en fases, se utilicen modelos con distinto nivel de detalle para reducir progresivamente el espacio de búsqueda y se implante con éxito el conocimiento de los expertos, si el problema de diseño es complicado el número de puntos evaluados seguirá siendo muy alto.

En este capítulo se ofrece una solución general para aumentar la velocidad del proceso. Se puede aplicar a cualquier función, ya sea escalar o vectorial, y a cualquier tipo de variables que intervengan en la función: continuas, discretas y muy discretas.

En el apartado 7.1 se presenta el uso de los ábacos como alternativa a los modelos simplificados de los procesos. En el 7.2 se describen los ábacos lineales de N dimensiones, y en el 7.3 el algoritmo seguido para su creación automática; en el apartado 7.4 se proponen los ábacos no lineales (o memorias selectivas) para casos con un número elevado de variables; finalmente, el apartado 7.5 está dedicado a los casos de aplicación de ábacos lineales que han aparecido en los proyectos en los que se basa esta tesis.

7.1 - Abacos y Modelos Simplificados.

Supongamos un proceso de cálculo (función) P tal que permite obtener el valor de una variable escalar **u** a partir de una variable vectorial X:

$$u = u(X) = u(x_1, x_2, ..., x_n)$$

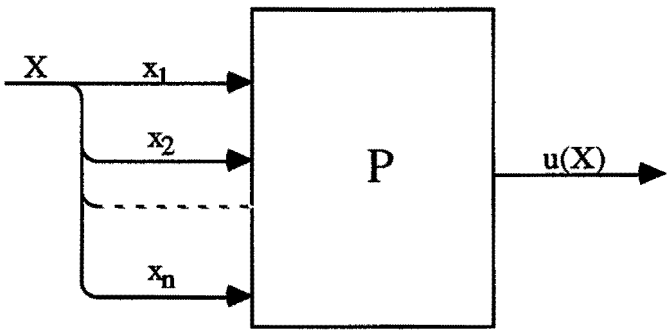


Figura 7.1 - Función escalar de variable vectorial.

Supóngase que el tiempo necesario para hallar **u** como función de X por ordenador resulte significativo, bien por sí mismo o por formar parte de un proceso iterativo que lo llama muchas veces. Para disminuir el tiempo de cálculo se puede sustituir el proceso **P**, que es la implantación de la función **u** (ver figura 7.1) por otro proceso equivalente **P*** que encuentre los valores de **u** en un tiempo menor a costa de producir un cierto error e(X) (ver figura 7.2)

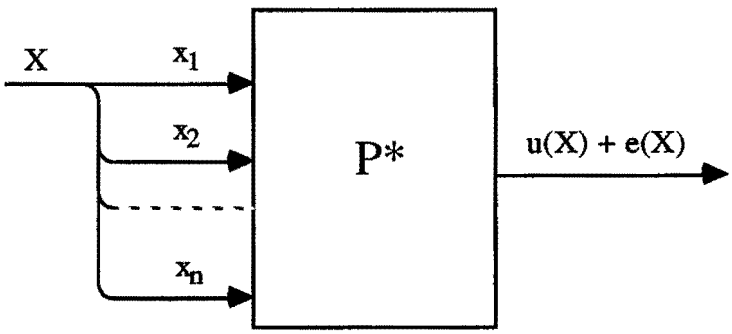


Figura 7.2 - Función rápida equivalente a P.

Un método utilizado tradicionalmente consiste en el desarrollo e implantación de un modelo simplificado de u(X) tal que simule su comportamiento dentro de unos márgenes de X pero mantenga el tiempo de cálculo y los errores dentro de unos niveles aceptables. El problema fundamental es que resulta difícil llegar a un compromiso entre tiempo y errores que sea válido para todo el rango de variación de X. Además, la reducción de tiempo no suele ser en absoluto espectacular, y la construcción de buenos modelos simplificados puede exigir estudios nada triviales.

7.1.1 - Planteamiento General.

La solución que aquí se propone como alternativa a los modelos simplificados es la creación y uso de ábacos de las funciones lentas. El **ábaco** de una función es otra función que tiene almacenado un conjunto de puntos significativos de la función original (**puntos de muestra**) y aproxima el valor del resto de los puntos por interpolación en n dimensiones.

Las **razones** principales que sugieren el uso de ábacos en vez de modelos simplificados son las siguientes:

- La lentitud de un proceso de cálculo P no implica en absoluto que la función u(X) correspondiente sea muy sensible ante variaciones locales de X. Antes bien, la mayoría de las funciones que representan fenómenos físicos reales (que son los que aparecen en la ingeniería) suelen ser suaves y de buen comportamiento local, al menos respecto a la mayoría de las componentes de X. Esto hace que sean fácilmente **interpolables**.

- Si el conjunto de puntos que se toma como muestra de u(X) para las interpolaciones es bueno, los **errores** resultantes de la **interpolación** son menores que los obtenidos con modelos simplificados.

- La única forma de aumentar la **precisión** en un modelo simplificado es aumentar también su complejidad, y por tanto su **lentitud**. Por desgracia, en los modelos simplificados el tiempo de cálculo tiende rápidamente al del proceso original al intentar aumentar su precisión. Además, el aumento de precisión logrado no es **uniforme** para todo el rango de variación de X. Sin embargo, en un ábaco, la precisión crece **localmente** en aquellas zonas del espacio definido por X en que se toman más puntos de muestra. El espacio de **memoria** necesario crece linealmente con el número de

puntos tomados de muestra, pero el aumento en el **tiempo** de cálculo respecto al número de puntos de muestra es despreciable (como ya se verá).

- Un ábaco no tiene ninguna dificultad en manejar variables **discretas** mezcladas con variables **continuas**, ya que por sí mismo maneja todas las variables como si fueran discretas, con incrementos de un cierto tamaño (ver apartado 5.1).

- Es imposible **programar** un "creador automático de modelos simplificados" que sirva para cualquier proceso P, pero es perfectamente posible hacer lo propio con un "creador automático de ábacos" porque los ábacos de distintas funciones presentan a una estructura común.

Un ábaco se considera más **eficaz** cuantos menos puntos necesite almacenar para realizar correctamente las interpolaciones. El número de puntos almacenados depende del número de variables de entrada, del tamaño de dichas variables y del comportamiento local de la función con respecto a ellas (ver apartado 5.1).

Se puede mejorar radicalmente la eficacia de un ábaco estudiando cuidadosamente la función lenta que se desea sustituir. La idea es aislar dentro de la función el proceso de cálculo realmente lento para sustituirlo por un ábaco. Es posible que el número de variables de entrada de dicho proceso sea menor que el de la función completa. Aparte de esto, hay que seleccionar las variables realmente importantes dentro del proceso lento, pues a veces con simples cambios de variable se mejora mucho el comportamiento local de la función resultante.

Sea un proceso **Q** cuyas variable vectorial de entrada es **Y** y cuya variable de salida es **v**. Sea $\mathbf{Q}^{\mathbf{a}}$ el proceso de interpolación en **n** dimensiones que nos permite simular el comportamiento de Q, es decir, Q^a es un ábaco de Q.

Un primer paso para que la implantación sea eficaz consiste en realizar un cambio de variable tal que:

- Se descompone el proceso en dos subprocesos (si es posible), uno lento y otro de cálculo inmediato, o sea compuesto de operaciones sencillas.

- Las variables de entrada del subproceso lento son las más importantes para dicho proceso.

En la figura 7.3 se representa la organización interna de un proceso ya descompuesto. El proceso Q^a de la figura sería el sustituto del Q original; el **interfaz** sería responsable del cambio de variables; P^a sería el ábaco del proceso lento y P^r el proceso rápido final ya mencionado que utiliza la variable intermedia producida por P^a.

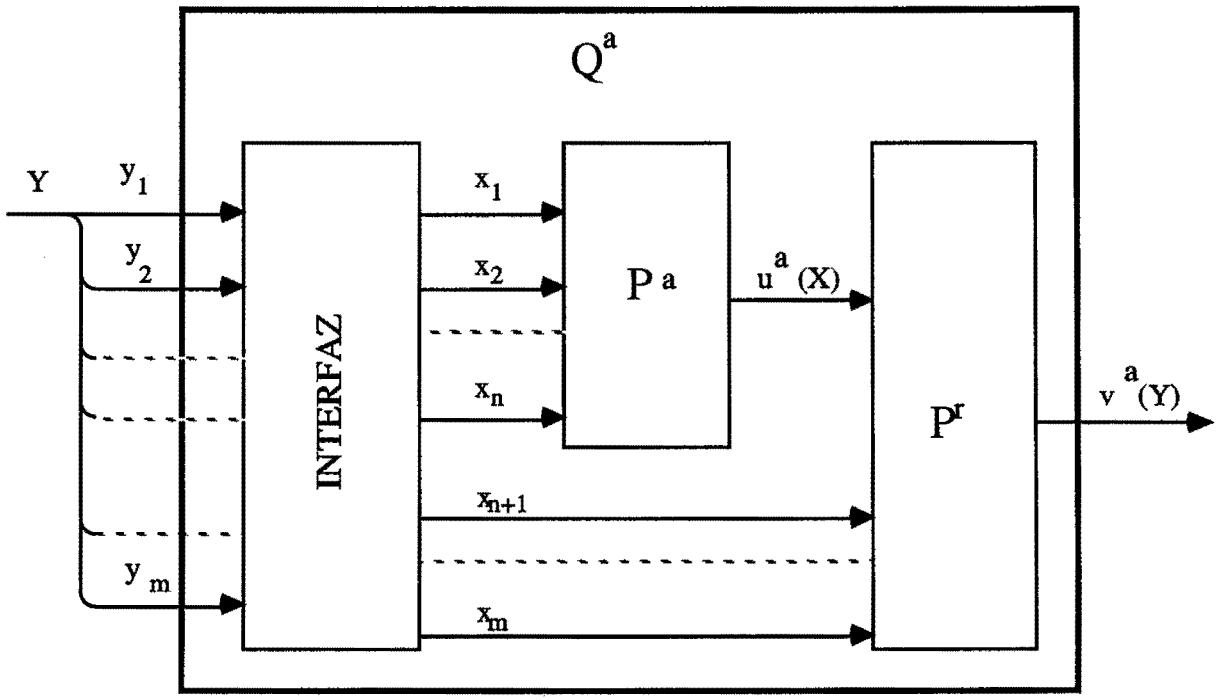


Figura 7.3. Descomposición de un proceso en procesos lentos y rápidos.

Naturalmente, en un caso general podrá haber más de un interfaz y más de un proceso rápido, dependiendo de los cambios de variables efectuados. En la figura aparece sólo un interfaz y un proceso rápido por no complicar innecesariamente el dibujo.

La descomposición del proceso es tarea de un experto que conozca bien la naturaleza del proceso Q, aunque una descomposición poco apropiada no significa peores resultados en la interpolación, sino una implantación menos eficaz.

7.2 - Abacos Lineales en N Dimensiones.

La interpolación lineal se suele asociar a funciones de una variable, y en ese contexto es la más sencilla y rápida de realizar. En este apartado se generaliza la interpolación lineal a un espacio de N variables.Se presenta también un algoritmo para encontrar un buen conjunto de puntos muestra de una función genérica (estrictamente no es un conjunto óptimo), y definir así un ábaco lineal de la función.

Este algoritmo se ha implantado con éxito, desarrollándose una herramienta que analiza sistemáticamente una función lenta ya programada y crea automáticamente una función rápida capaz de sustituirla con un margen de error prefijado, o sea, escribe el código del ábaco de una función dada.

7.2.1 - Función de Interpolación Lineal en N Dimensiones.

Sea una función escalar **u** definida en un espacio de **n** dimensiones:

$$u(X) = u(x_1, x_2, ..., x_n)$$

Supóngase que no se conoce el valor de $u(X)$ para cualquier valor de x_1, sino sólo para un conjunto ordenado Mx_1 de m_1 valores de muestra:

$$Mx_1 = \{ mx_{1(1)}, mx_{1(2)}, ..., mx_{1(m_1)} \}$$

tal que:

$$mx_{1(i_1)} < mx_{1(i_1+1)} \qquad \forall\ i_1 \in \{ 1, 2,..., m_1 - 1 \}$$

donde $mx_{1(i_1)}$ debe leerse como "valor de muestra número i_1 de la variable x_1".

Aunque x_1 no coincida con ninguno de los valores $mx_{1(i_1)}$ de muestra, se podrá obtener una aproximación de $u(X)$ mediante una interpolación lineal entre los dos valores conocidos de $u(X)$ que estén más próximos, es decir:

Dados:

$$mx_{1(i_1)}, mx_{1(i_1+1)}$$

tales que:

$$mx_1(i_1) < x_1 < mx_1(i_1+1)$$

se puede obtener una aproximación $u^a(X)$ de $u(X)$ por medio de una interpolación lineal en una dimensión entre el valor más próximo por defecto:

$$u(mx_1(i_1), x_2, ..., x_n)$$

y el valor más próximo por exceso:

$$u(mx_1(i_1+1), x_2, ..., x_n)$$

El resultado de esta interpolación lineal en una dimensión será:

$$u^a(X) = u(mx_1(i_1), x_2, ..., x_n) + \frac{x_1 - mx_1(i_1)}{mx_1(i_1+1) - mx_1(i_1)} \left[u(mx_1(i_1+1), x_2, ..., x_n) - u(mx_1(i_1), x_2, ..., x_n) \right]$$

que escrito de forma más compacta queda:

$$u^a(X) = p_0(i_1)\ u(mx_1(i_1), x_2, ..., x_n) + p_1(i_1)\ u(mx_1(i_1+1), x_2, ..., x_n)$$

siendo:

$$p_0(i_1) = \frac{mx_1(i_1+1) - x_1}{mx_1(i_1+1) - mx_1(i_1)} = 1 - p_1(i_1)$$

$$p_1(i_1) = \frac{x_1 - mx_1(i_1)}{mx_1(i_1+1) - mx_1(i_1)} = 1 - p_0(i_1)$$

Los números adimensionales $\mathbf{p_0(i_1)}$ y $\mathbf{p_1(i_1)}$ son las distancias unitarias de x_1 a los extremos del intervalo $[mx_1(i_1), mx_1(i_1+1)]$, relativas a la longitud de dicho intervalo.

Para poder realizar la interpolación se necesita conocer el valor de la función $u(X)$ en ambos extremos del intervalo. Pero supóngase ahora que tampoco se conoce $u(X)$ para cualquier valor de x_2, sino sólo para un conjunto ordenado Mx_2 de m_2 valores de muestra:

$$Mx_2 = \{ mx_2(1), mx_2(2), ..., mx_2(m_2) \}$$

tal que:

$$mx_{2(i_2)} < mx_{2(i_2+1)} \qquad \forall\ i_2 \in \{ 1, 2, ..., m_2 - 1 \}$$

donde $mx_{2(i_2)}$ debe leerse como "valor de muestra número i_2 de la variable x_2".

En este caso los dos valores de la función necesarios para la interpolación en x_1 se pueden aproximar a su vez por medio de sendas interpolaciones en una dimensión:

$$u^a(mx_{1(i_1)}, x_2, ..., x_n) = p_{0(i_2)}\ u(mx_{1(i_1)}, mx_{2(i_2)}, ..., x_n) + p_{1(i_2)}\ u(mx_{1(i_1)}, mx_{2(i_2+1)}, ..., x_n)$$

$$u^a(mx_{1(i_1+1)}, x_2, ..., x_n) = p_{0(i_2)}\ u(mx_{1(i_1+1)}, mx_{2(i_2)}, ..., x_n) + p_{1(i_2)}\ u(mx_{1(i_1+1)}, mx_{2(i_2+1)}, ..., x_n)$$

siendo:

$$p_{0(i_2)} = \frac{mx_{2(i_2+1)} - x_2}{mx_{2(i_2+1)} - mx_{2(i_2)}} = 1 - p_{1(i_2)}$$

$$p_{1(i_2)} = \frac{x_2 - mx_{2(i_2)}}{mx_{2(i_2+1)} - mx_{2(i_2)}} = 1 - p_{0(i_2)}$$

y donde los números adimensionales $p_{0(i_2)}$ y $p_{1(i_2)}$ son las distancias unitarias de x_2 a los extremos del intervalo $[mx_{2(i_2)}, mx_{2(i_2+1)}]$, relativas a la longitud de dicho intervalo.

Así se podría seguir recursivamente hasta interpolar en x_n. Se demuestra fácilmente por inducción que la **expresión analítica de la función interpolación en n dimensiones** queda:

$$u^a(x_1, x_2, ..., x_n) =$$

$$\sum_{k1=0}^{1} .... \sum_{kn=0}^{1} \left[u(mx_{1(i_1+k1)}, mx_{2(i_2+k2)}, ..., mx_{n(i_n+kn)}) \prod_{j=1}^{n} p_{kj(i_j)} \right]$$

siendo:

$$p_{0(i_j)} = \frac{mx_{j(i_j+1)} - x_j}{mx_{j(i_j+1)} - mx_{j(i_j)}} = 1 - p_{1(i_j)}$$

$$p_{1(i_j)} = \frac{x_j - mx_{j(i_j)}}{mx_{j(i_j+1)} - mx_{j(i_j)}} = 1 - p_{0(i_j)}$$

y siendo:

$$mx_{j(i_j)} < mx_{j(i_j+1)} \qquad \forall\ j \in \{ 1, 2, ..., n \}$$

La función de interpolación lineal así definida presenta las siguientes **propiedades:**

- Es continua.
- Pasa por todos los puntos muestra de la función original
- En el centro de cada hiperintervalo vale la media aritmética de los puntos muestra que lo delimitan.
- En una variable es una interpolación lineal convencional (ver figura 7.4).

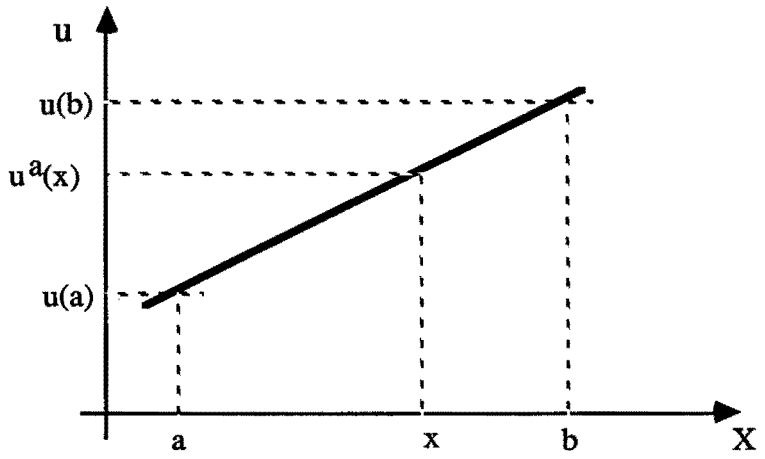


Figura 7.4. Interpolación Lineal en una dimensión

- En los puntos de las aristas de los hiperintervalos se hallan los valores de $u^a(X)$ con interpolaciones lineales de una sola variable.

- En dos variables es una superficie formada por superficies regladas que se apoyan en los cuatro segmentos que unen los puntos muestra de cada hiperintervalo (ver figura 7.5)

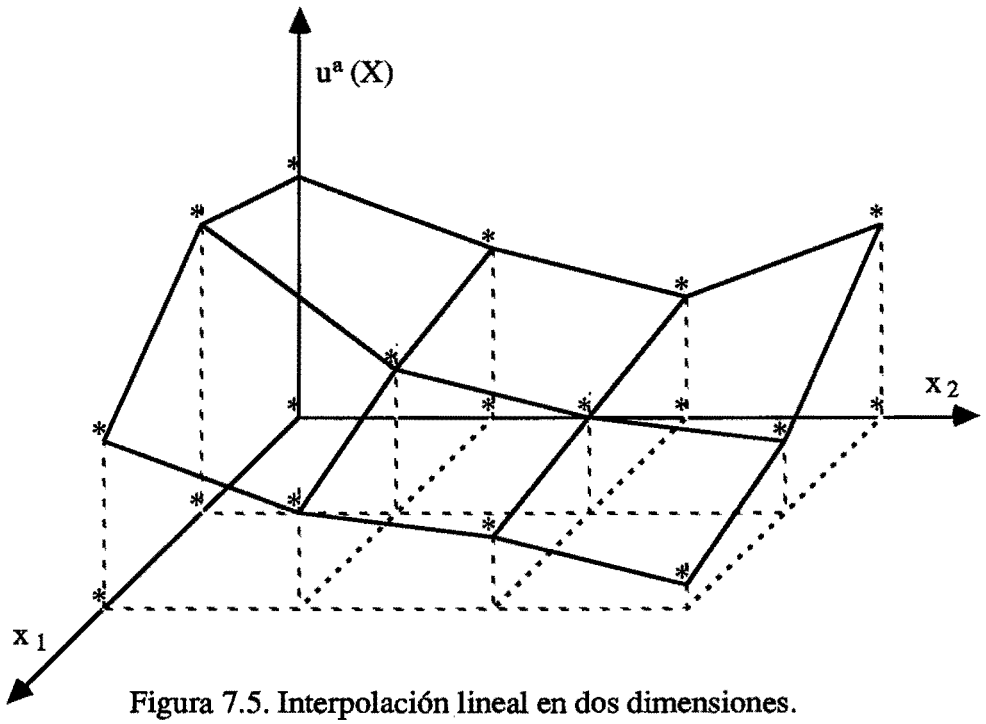


Figura 7.5. Interpolación lineal en dos dimensiones.

Para poder realizar la interpolación es necesario conocer:

- m_j Número de puntos de muestra de cada variable x_j

- Mx_j Vector de puntos de muestra de cada variable x_j
  $Mx_j = \{ mx_{j(1)}, mx_{j(2)}, ..., mx_{j(m_j)} \}$

- Mu(X) Matriz de valores de muestra de u(X). Son los valores reales de la función u(X) en todos los puntos muestra tomados:

  $Mu(X) = (mx_{1(i_1)}, mx_{2(i_2)}, ..., mx_{n(i_n)})$

  Habrá un total de puntos almacenados en Mu(X) igual a:

$$n^{o} \text{ de puntos} = \prod_{j=1}^{n} m_j$$

El proceso de interpolación consta de dos fases:

- **Localización:**

  Conociendo la muestra Mx_j de cada variable x_j
  Conociendo el punto $X = (x_1, x_2, ..., x_n)$

  Hay que hallar el conjunto de enteros

  $$I = \{ i_1, i_2, ..., i_n \}$$

  tales que:

  $$mx_{j(i_j)} \leq x_j \leq mx_{j(i_j+1)} \qquad \forall\ j \in \{ 1, 2, ..., n \}$$

- **Cálculo:** Consiste en aplicar la función de interpolación lineal en N dimensiones que ya se ha explicado.

Para aclarar el proceso y la función de interpolación presentados lo mejor es seguir paso a paso un ejemplo en dos variables. Sea una función de variable vectorial u(X):

$$u(X) = u(x_1, x_2)$$

definida en la región:

$$0 \leq x_1 \leq 9$$
$$2 \leq x_2 \leq 8$$

y cuya ley de obtención de imágenes es:

$$u(X) = \sqrt{x_1}\ x_2$$

De esta función sólo se conocen los valores correspondientes a los siguientes puntos de muestra de las variables:

$$Mx_1 = \{ 0, 5, 9 \} \qquad m_1 = 3$$
$$Mx_2 = \{ 2, 8 \} \qquad m_2 = 2$$

Por tanto la matriz de valores de muestra de u(X) será:

$$Mu(X) = \begin{bmatrix} u(0,2) & u(0,8) \\ u(5,2) & u(5,8) \\ u(9,2) & u(9,8) \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ 2\sqrt{5} & 8\sqrt{5} \\ 6 & 24 \end{bmatrix}$$

Si ahora se desea hallar el valor de la función u(X) en el punto (6,4) por medio de una interpolación lineal en dos dimensiones habrá que seguir los pasos:

- **Localización:**

Conociendo la muestra Mx_j de cada variable x_j

$$Mx_1 = \{ 0, 5, 9 \} = \{ mx_{1(1)}, mx_{1(2)}, mx_{1(3)} \}$$
$$Mx_2 = \{ 2, 8 \} = \{ mx_{2(1)}, mx_{2(2)} \}$$

Conociendo el punto $X = (6, 4) = (x_1, x_2)$

Sale el conjunto de enteros:

$$I = \{ i_1, i_2 \} = \{ 2, 1 \}$$

porque:

$$mx_{1(2)} \leq 6 \leq mx_{1(2+1)}$$
$$mx_{2(1)} \leq 4 \leq mx_{2(1+1)}$$

- **Cálculo:**

Primero se hallan las distancias relativas:

$$p_0(i_1) = \frac{mx_{1(3)} - x_1}{mx_{1(3)} - mx_{1(2)}} = \frac{9 - 6}{9 - 5} = \frac{3}{4}$$

$$p_1(i_1) = \frac{x_1 - mx_{1(2)}}{mx_{1(3)} - mx_{1(2)}} = \frac{6 - 5}{9 - 5} = \frac{1}{4}$$

$$p_0(i_2) = \frac{mx_{2(2)} - x_2}{mx_{2(2)} - mx_{2(1)}} = \frac{8 - 4}{8 - 2} = \frac{2}{3}$$

$$p_1(i_2) = \frac{x_2 - mx_{2(1)}}{mx_{2(2)} - mx_{2(1)}} = \frac{4 - 2}{8 - 2} = \frac{1}{3}$$

Y ya se puede aplicar la fórmula recursiva de interpolación lineal:

$$u^a(x_1, x_2) = \sum_{k1=0}^{1} \sum_{k2=0}^{1} \left[Mu(2+k1, 1+k2) \prod_{j=1}^{2} p_{kj(ij)} \right]$$

$$u^a(6, 4) = Mu(2,1)\ \frac{3}{4}\frac{2}{3} + Mu(2,2)\ \frac{3}{4}\frac{1}{3} + Mu(3,1)\ \frac{1}{4}\frac{2}{3} + Mu(3,2)\ \frac{1}{4}\frac{1}{3} =$$

$$= 2\sqrt{5}\ \frac{1}{2} + 8\sqrt{5}\ \frac{1}{4} + 6\ \frac{1}{6} + 24\ \frac{1}{12} = 9.7082$$

Se puede comparar ahora el valor obtenido con el valor real de la función u(X) en el punto en cuestión, que es:

$$u(6, 4) = \sqrt{6}\ 4 = 9.7979$$

7.3 - Algoritmo de Creación Automática de Abacos Lineales.

El algoritmo está basado en la selección adecuada de los puntos muestra. Es fácilmente observable en funciones de una variable que para realizar una interpolación lineal es más importante la selección de los puntos muestra que el número de los mismos. La figura 7.6 compara una elección adecuada de puntos con otra poco adecuada.

Las funciones definidas en espacios de n dimensiones serán en general irrepresentables gráficamente. Sin embargo se puede desarrollar un algoritmo sistemático para seleccionar puntos muestra adecuados para una función genérica definida en un espacio genérico, aunque limitando el dominio de la función a un hiperintervalo concreto. Lo que se pretende es encontrar un conjunto mínimo de puntos muestra tal que la interpolación lineal a partir de dichos puntos produzca resultados dentro de un margen de error prefijado para todo punto de dicho hiperintervalo.

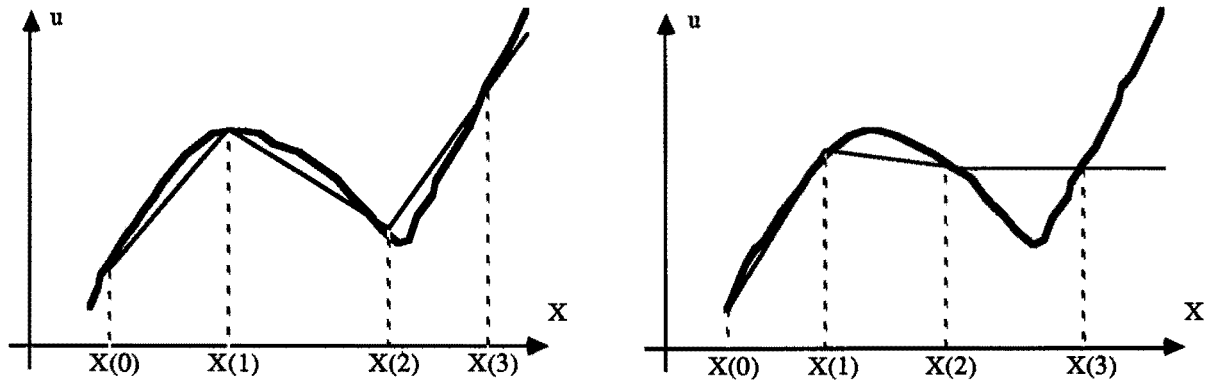


Figura 7.6. Importancia de una elección adecuada de puntos muestra.

Se intenta crear un ábaco de una función de evaluación lenta, para poder sustituirla por aquél incurriendo en errores admisibles pero aumentando espectacularmente la rapidez de evaluación. Si la función es lenta, también la construcción de su ábaco lo será; pero esto sólo debe hacerse una vez, mientras que si la función forma parte de un proceso iterativo los beneficios obtenidos con la sustitución son muy importantes. Este es el caso del proceso de evaluación de un diseño como parte de un algoritmo de búsqueda del diseño óptimo.

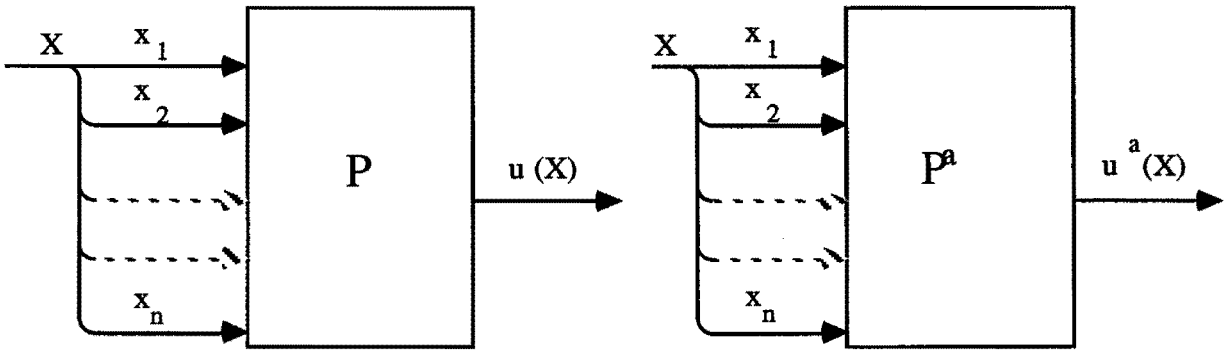


Figura 7.7. Procesos cuyos resultados se comparan para comprobar la precisión.

El algoritmo compara sistemáticamente los valores obtenidos mediante los dos procesos de la figura 7.7. P es el proceso lento que da los valores reales de la función u(X), mientras que P^a es el ábaco provisional de dicho proceso que calcula los valores de la función aproximada

$u^a(X)$. Según aumente el número de puntos almacenados en P^a, las diferencias o errores entre ambos procesos irán reduciéndose. Cuando los errores estén dentro de los límites prefijados se dará por concluido el proceso.

En el apartado 7.3.1 se describirá brevemente el algoritmo propuesto para una función de una variable. En el apartado 7.3.2 se generalizará el algoritmo para espacios de n variables, y en el 7.3.3 se mostrará la estructura interna de un ábaco genérico.

7.3.1 - Algoritmo en Una Dimensión.

Sea una función u(x) de cálculo lento, de la cual se quiere obtener un ábaco que la sustituya con un error máximo admisible ε , dentro de un intervalo de definición $[x(0),x(1)]$

El ábaco provisional inicial cuenta sólo con los extremos del intervalo como puntos muestra. La figura 7.8 muestra la comparación entre la función real u(x) y el ábaco provisional inicial $u^a(x)$.

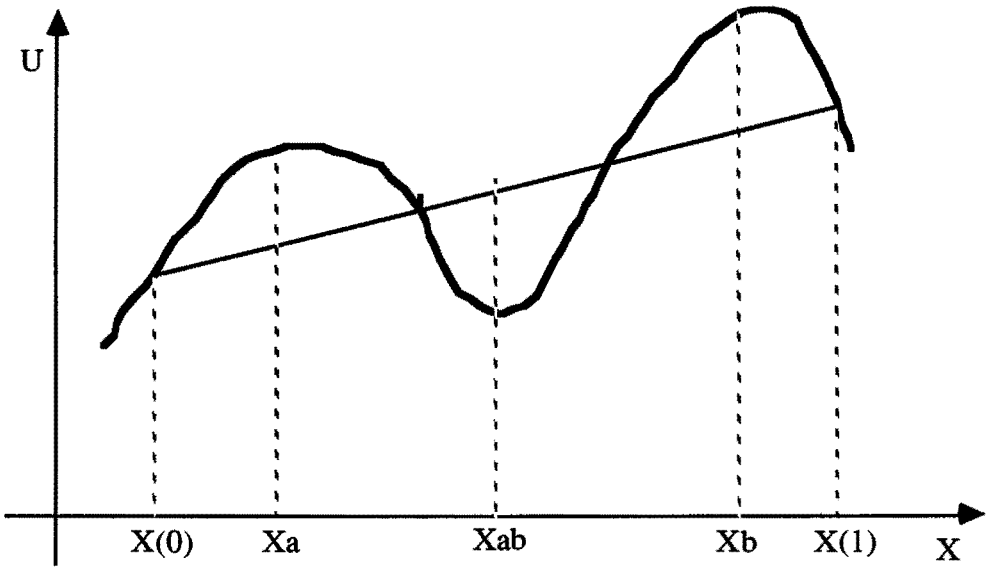


Figura 7.8. Interpolación lineal entre los extremos del intervalo de definición.

Se intenta ahora encontrar el punto más adecuado para ser añadido a la lista de puntos muestra. El criterio que se sigue aquí es seleccionar el punto que provisionalmente tenga el máximo error, o sea:

Hallar x tal que $|\varepsilon(x)|$ sea máximo.
siendo $\varepsilon(x) = u(x) - u^a(x)$

Para ello se deben realizar optimizaciones (maximizaciones) partiendo de distintos puntos iniciales, dado el posible mal comportamiento local de la función u(x). En la figura 7.8 se muestra el resultado de tres maximizaciones, tomando los dos extremos del intervalo y el punto medio como puntos iniciales.

En general cada optimización llegará a un máximo local distinto. Si en todos ellos el error es menor que el máximo admisible, el ábaco está ya terminado. Si no es así, se selecciona el punto de mayor error y se incorpora a la lista de puntos muestra.

Supóngase que el punto de mayor error en la figura 7.8 sea el x_a. La lista de puntos muestra del ábaco provisional tendría tres elementos, y la función $u^a(x)$ sería un poco más precisa. La figura 7.9 muestra la nueva función $u^a(x)$ comparada con la función u(x) original.

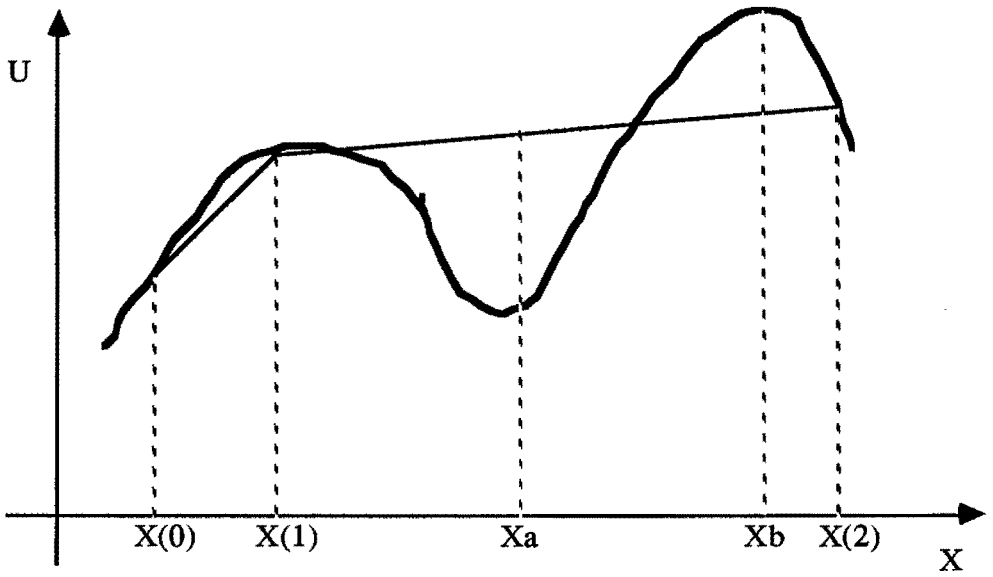


Figura 7.9. Interpolación lineal con un punto muestra adicional.

El dominio de la función queda partido en dos intervalos, $[x_{(0)},x_{(1)}]$ y $[x_{(1)},x_{(2)}]$, según muestra la figura 7.9. A cada uno de ellos se le aplicará el mismo proceso de búsqueda de error máximo, hasta que en ningún intervalo se alcance un error mayor que el máximo admisible.

En la figura 7.9 se supone que el primer intervalo $[x_{(0)},x_{(1)}]$ ya es suficientemente preciso, mientras que en el segundo aparecen dos máximos locales de error. La figura 7.10 muestra dos pasos más del proceso, hasta quedar la función u(x) aproximada con suficiente precisión por medio de cinco puntos muestra.

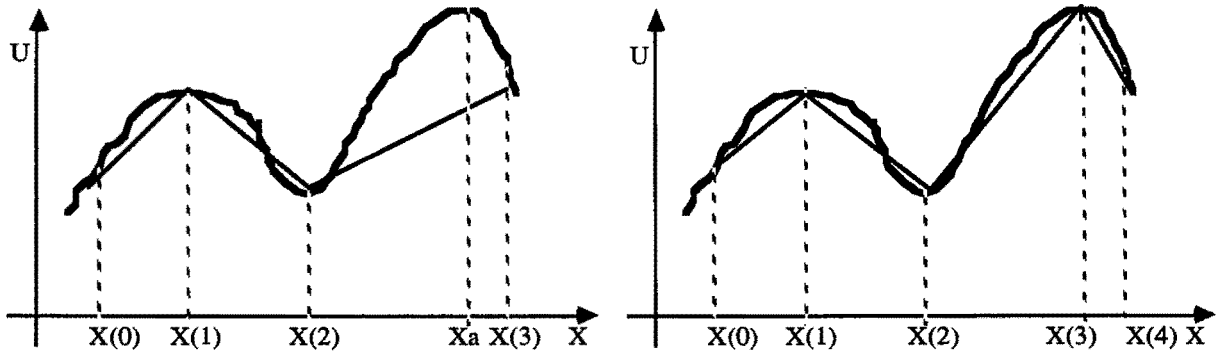


Figura 7.10. Adición de nuevos puntos muestra.

7.3.2 - Algoritmo en N Dimensiones.

Al dar el salto a funciones definidas en espacios de más de una dimensión el algoritmo se complica, aunque siga la misma filosofía. Se comentarán aquí las características más interesantes que diferencian el algoritmo en n dimensiones del algoritmo descrito en el apartado anterior, sin entrar en descripciones detalladas.

En vez de trabajar con intervalos definidos por 2 puntos, se trabaja con hiperintervalos definidos por 2^n puntos. Por ejemplo, el hiperintervalo inicial de definición de la función será:

$$[X_{(0)},X_{(1)}] = \{ \; x_1(0) \le x_1 \le x_1(1) \; ; \; x_2(0) \le x_2 \le x_2(1) \; ; \; \ldots \; ; \; x_n(0) \le x_n \le x_n(1) \; \}$$

Un ábaco provisional no será una lista de puntos, sino una malla irregular de puntos muestra que define un conjunto de hiperintervalos organizados en forma matricial. Para buscar puntos de máximo error se examinan las aristas de los hiperintervalos y sus interiores. Cada vez que se localiza un nuevo punto de máximo error se genera todo un conjunto de puntos muestra, al proyectar el nuevo punto en los n ejes e intersecar con la malla provisional existente.

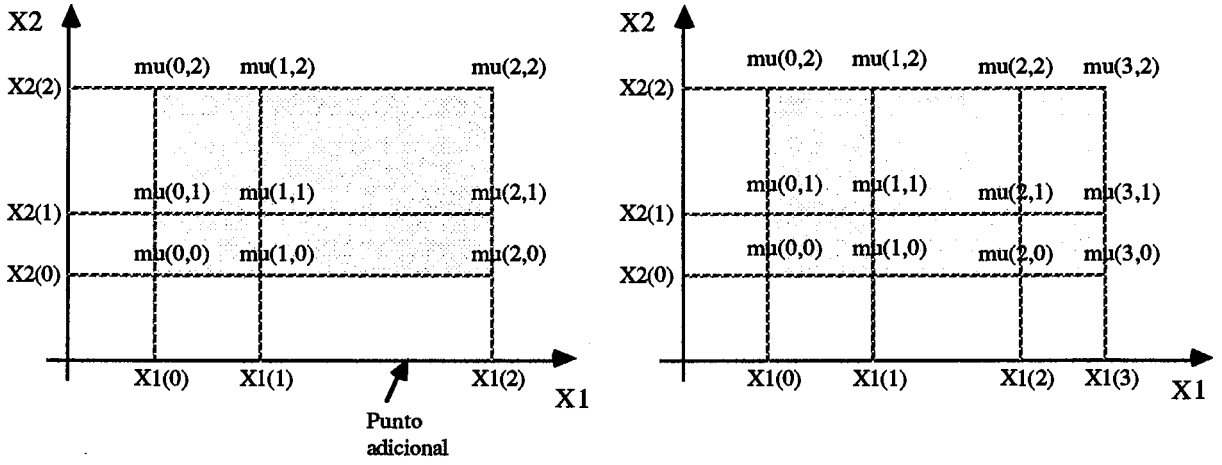


Figura 7.11. Adición de un punto en un eje de la malla de puntos de un ábaco en dos dimensiones.

En la figura 7.11 aparece una malla de puntos muestra de un ábaco provisional en dos dimensiones, y cómo se altera con la adición de un nuevo punto muestra en un eje. La explosión combinatoria de puntos muestra puede resultar en un número de puntos inadmisible. La solución adoptada aquí consiste en aislar los hiperintervalos con error y estudiarlos con más detalle, es decir, con una malla más fina.de hiperintervalos. La figura 7.12 muestra el resultado de este tipo de mallado selectivo.

Cuando se desee interpolar el valor de la función en un punto dado, se empleará la malla de puntos muestra correspondiente al hiperintervalo al que pertenezca dicho punto, es decir, se empleará el ábaco correspondiente a dicho punto. Lo que se obtiene con un mallado a distintos niveles de detalle es un árbol de ábacos jerarquizados, cuya estructura interna se muestra en el siguiente apartado.

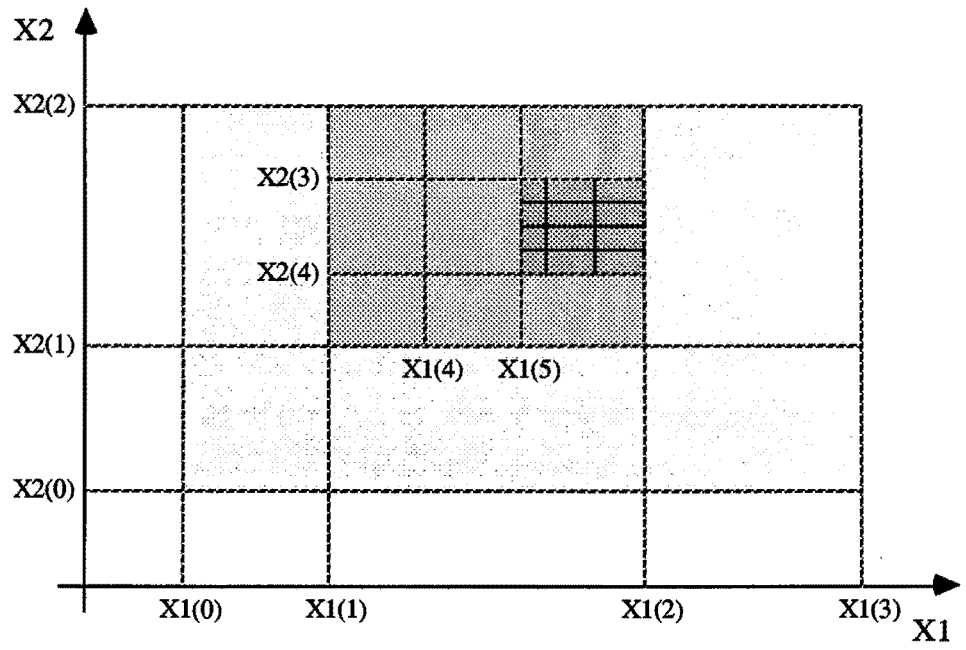


Figura 7.12. La malla de puntos muestra es más fina en zonas conflictivas de la función.

7.3.3 - Estructura de un Abaco Lineal.

La figura 7.13 muestra la estructura resultante de un ábaco de interpolación lineal en n dimensiones. Junto al interfaz y el proceso de cálculo rápido responsables de los cambios de variable (ver figura 7.3 en el apartado 7.1), aparece el **ábaco principal P^a**.

El ábaco principal se compone de un localizador y de un interpolador. El **localizador** tiene que determinar para cada nuevo punto X el hiperintervalo que le corresponde, es decir, el hiperintervalo $[X_{(i)},X_{(i+1)}]$ que cumple que:

$$X_{(i)} \leq X \leq X_{(i+1)}$$

Una vez localizado el hiperintervalo, el **interpolador** tiene que calcular el valor aproximado de la función $U^a(X)$. Si el hiperintervalo es tal que con la malla del ábaco principal se puede interpolar la función con poco error, el **calculador** se encarga de aplicar la función de interpolación lineal en n dimensiones. Si no es así, el interpolador llama a otro ábaco (un ábaco secundario) definido en dicho hiperintervalo y que tiene una malla más fina de puntos muestra.

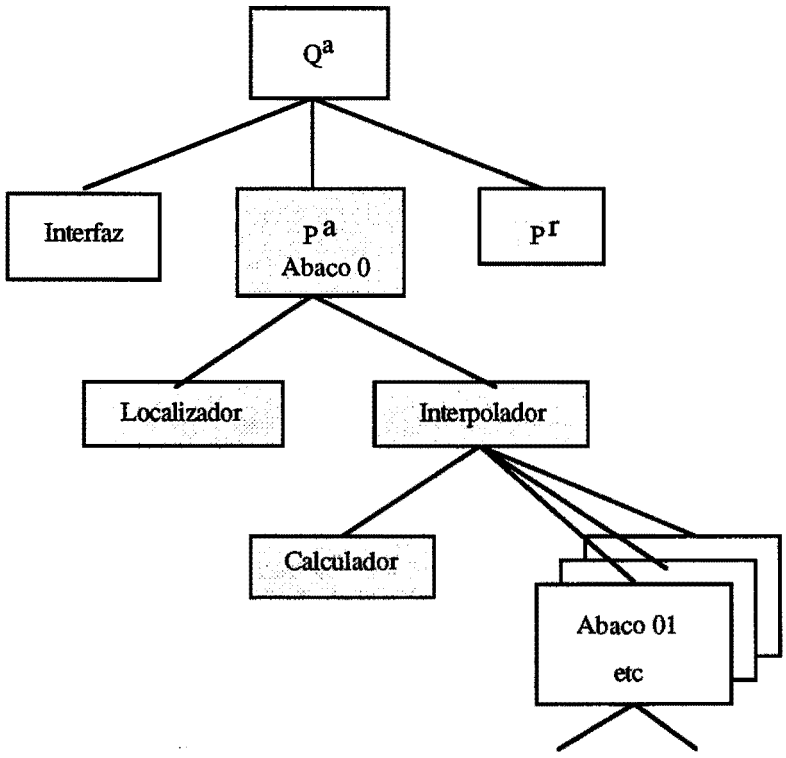


Figura 7.13. Estructura interna de un ábaco lineal en N dimensiones.

Cada ábaco secundario presenta la misma estructura del ábaco principal; tendrá un localizador para su propia malla y un interpolador, que a su vez podrá llamar a otros ábacos secundarios de nivel inferior, y por tanto con una malla de puntos aún más fina. La estructura se repite a sí misma recursivamente, hasta que en todos los hiperintervalos se pueda interpolar con un error admisible.

7.4 - Abacos No Lineales en N Dimensiones: Memorias Selectivas.

Los ábacos lineales son muy útiles pero tienen ciertas limitaciones: la principal de ellas es el número de variables del espacio en el que está definida la función a sustituir. Basta darse cuenta de que el número de puntos que es necesario conocer para interpolar en cada intervalo es 2^n, lo que hace que no sea práctico crear ábacos en espacios de más de tres o cuatro dimensiones.

En este apartado se propone cómo afrontar la creación de ábacos de funciones definidas en espacios de muchas variables. El sistema está basado en el almacenamiento de puntos por **memorias selectivas**, y es una aportación original de esta tesis que no ha sido utilizada en proyectos reales, sino sólo en casos ejemplo.

Las características de un ábaco genérico son:

i) Es una función capaz de sustituir a otra función más lenta, a costa de un cierto margen de error.

ii) Los resultados se hallan por interpolación entre los valores de un conjunto de puntos muestra previamente almacenados.

iii) Se crea dinámicamente, añadiendo puntos muestra nuevos en las zonas donde el error sea mayor que el admisible.

Los ábacos lineales utilizan una buena función de interpolación y un sistema muy rápido de localización de los puntos muestra utilizados en la misma; como contrapartida, exigen una distribución en forma de malla de los puntos muestra, lo que limita mucho el número máximo de variables. En ábacos no lineales la distribución de los puntos muestra podrá ser aleatoria, dificultando la tarea de localizar los puntos muestra más próximos a un punto dado y haciendo menos precisa la función de interpolación.

Se cuenta pues con un espacio de gran número de dimensiones. Se supone que no es posible aplicar los principios del **espacio heurístico** para reducir el número de variables (ver apartado 5.3.3) o bien que una vez aplicados todavía quedan demasiadas variables como para crear un ábaco lineal. Para poder implantar un sistema de memorias selectivas hace falta:

- Definir una función **distancia** entre los puntos del espacio tal que la proximidad según dicha distancia implique un valor parecido en la función que se desea sustituir.

- Definir una **función de interpolación** que tenga en cuenta los puntos más próximos al dado, según la función distancia seleccionada.

- Definir un **margen de error** máximo admisible entre el valor interpolado y la función real.

- Determinar un **algoritmo de ordenación** de puntos basado en la distancia, de tal manera que se pueda seleccionar rápidamente el grupo de puntos muestra más próximos a un punto dado.

El sistema de memorias selectivas funciona como sigue: dado un conjunto provisional de puntos muestra y un punto que se desea evaluar rápidamente, se selecciona (según el algoritmo de ordenación) un conjunto de puntos muestra que estén suficientemente próximos a dicho punto, según la función distancia definida. A continuación se halla el valor aproximado de la función en dicho punto según la función de interpolación definida, ponderándose más el valor de los puntos muestra más próximos.

Si se desea **aumentar el número de puntos muestra** para aumentar la precisión, se compara el valor aproximado de la función en cada nuevo punto con el valor real; si el error es mayor que el máximo admisible, se incorpora dicho punto al conjunto de puntos muestra según el algoritmo de ordenación utilizado. De esta manera sólo se guardan en memoria aquellos puntos que añaden información significativa sobre la función. El efecto que se desea conseguir es el de concentrar más puntos en las zonas de mal comportamiento local de la función y menos en las de buen comportamiento.

El algoritmo de ordenación que aquí se propone consiste en una organización dinámica en **red de árboles** basada en una distancia definida entre puntos. Cada nodo de un árbol es un punto que lleva asociado un entorno, definido en función de la distancia utilizada: todos los puntos situados dentro de ese entorno "colgarán" de dicho punto en forma de nodos hijo. El árbol crecerá tanto hacia arriba como hacia abajo según las distancias de los nuevos puntos con respecto a los antiguos y se organizará dinámicamente: los puntos podrán cambiar de nivel y sus distancias asociadas podrán actualizarse.

La figura 7.14 muestra un conjunto de puntos distribuidos irregularmente en un espacio de dos dimensiones; cada punto lleva asociado un entorno, que en el ejemplo de la figura es un círculo centrado en el punto porque se utiliza la distancia euclídea. La figura muestra también una red de árboles asociada a dicho sistema de puntos.

Para localizar los N puntos más próximos a uno dado se seleccionan primero los de más alto nivel en el árbol, y se confecciona una primera lista. Se comprueban a continuación las distancias a los puntos que cuelgan del más próximo, actualizándose la lista de N puntos, y así sucesivamente. El objetivo es comprobar la distancia a un número reducido de puntos gracias a

la ramificación. Ciertas ramas del árbol se pueden descartar a priori, pues se conocen las coordenadas del punto representativo de cada rama y el radio máximo del entorno que dicho punto cubre (todos los puntos que dependen del punto representativo deben estar dentro del entorno correspondiente).

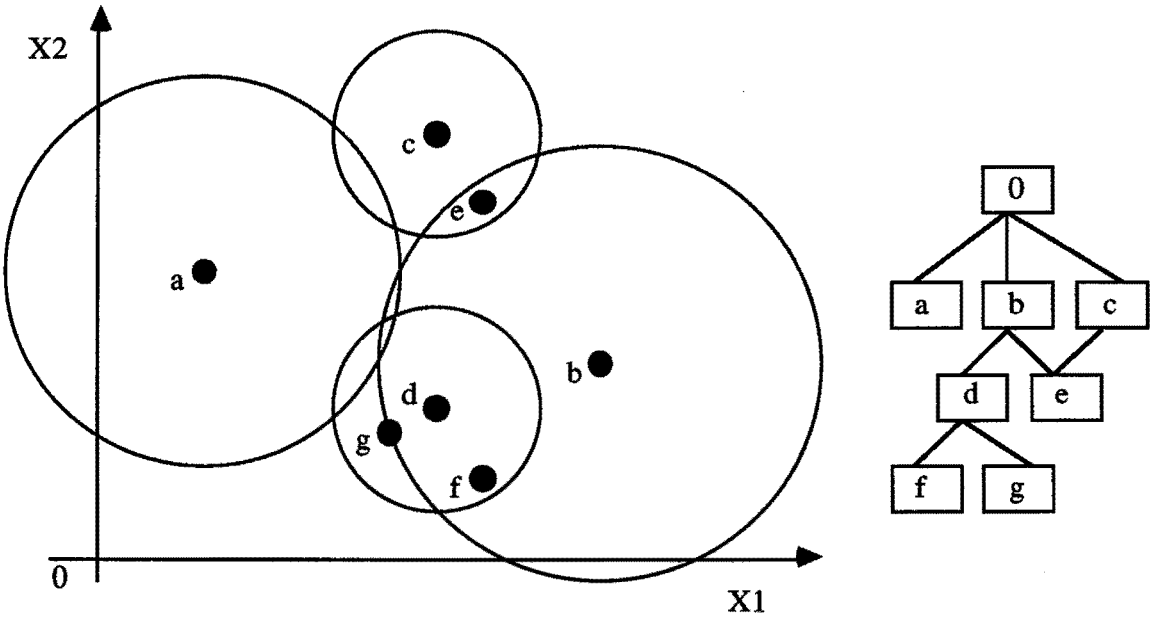


Figura 7.14. Organización de un conjunto de puntos en red de árboles según la distancia euclídea.

A continuación se expone un ejemplo basado en la figura 7.14. Supóngase que el punto "g" de la figura no estuviera almacenado en la red de puntos, y se deseara encontrar los 2 puntos más próximos a "g" para interpolar el valor de una cierta función F(X) en el punto "g" conociendo los valores de F(X) en todos los puntos de la red. Habría que seguir los pasos siguientes:

i) Se buscan los dos puntos más próximos a "g" entre los puntos de nivel superior {a,b,c}. Los dos más cercanos en este ejemplo son {a,b}, y la rama de "c" se puede despreciar pues según el radio de su entorno ningún punto que dependa de "c" puede desbancar ni a "a" ni a "b" de la lista.

ii) De entre {a,b} se selecciona el más próximo, que resulta ser "b", y se estudian los puntos de su entorno. El punto "e" se desprecia, por estar más lejos que "a" (que es el peor de la lista provisional) y no tener descendientes. El punto "d" sustituye al "a" en la

lista provisional, pero la rama "a" no se desprecia todavía, pues algún punto dependiente de "a" (el entorno de "a" tiene un radio muy grande) podría estar más cerca que "b". La nueva lista provisional es {d,b}.

iii) Se estudian los puntos del entorno de "d" (el punto más próximo de la lista) y como consecuencia el punto "f" desbanca a "b". Por último se comprueba la rama "a" que quedó abierta, y al no tener puntos en su entorno se acaba el proceso, con el resultado {f,d}.

iv) Se calcula ahora el valor aproximado de F(g) a partir de los valores {F(f),F(d)}.

v) Si ahora se plantease el incorporar o no el punto "g" al árbol de puntos muestra, se compararía el valor interpolado de F(g) con el valor real. Si el error fuera pequeño, "g" no se incorporaría al árbol de puntos. Si el error fuera grande, el punto "g" se incorporaría al árbol, pasando a depender del punto "d" por estar dentro de su entorno; también se debería comprobar si el nuevo punto debe depender también de otros puntos ya existentes en el árbol.

Así descrito, el proceso puede parecer muy ineficaz, pero debe comprenderse que el interés del método radica en la posibilidad -no muy explotada en el ejemplo- de despreciar ramas completas del árbol (y con ellas bloques enteros de puntos), en la generalización a cualquier número de dimensiones y en la generalización a cualquier tipo de distancia. Por otro lado, la eficacia de la búsqueda depende en gran medida de la estrategia seguida al crear dinámicamente el árbol de puntos muestra.

La construcción dinámica del árbol puede controlarse por medio de distintos heurísticos. Se puede controlar por distancias, por número máximo de hijos, etc. Cada vez que se incorpora un punto nuevo se intentará situar colgando de uno existente. Si esto no es posible, el nuevo punto se situará en la zona más alta del árbol, al primer nivel de ramificación. Es así como crece el árbol tanto hacia arriba como hacia abajo.

Este tipo de organización permite el aprendizaje y, por tanto, permite extraer información sobre el comportamiento de una función en las zonas de más interés de la misma. Si se desea construir un ábaco sistemáticamente, se puede generar aleatoriamente un conjunto de puntos muy alto y almacenar selectivamente sólo los que aporten información útil, es decir, los puntos cuyo valor no puede ser interpolado a partir de los puntos muestra ya existentes.

7.5 - Notas sobre Aplicaciones a Proyectos.

Las ideas y metodologías expuestas en este capítulo están inspiradas en el trabajo llevado a cabo en proyectos reales de investigación y desarrollo. En este apartado se comenta el trabajo realizado en dichos proyectos a la luz de las ideas y la nomenclatura utilizadas aquí, como si éstos fueran consecuencia de la tesis y no al contrario. De esta forma las aplicaciones reales proporcionan un respaldo experimental a las ideas y metodologías expuestas en la tesis.

La necesidad de un método de interpolación lineal de funciones de varias variables se hizo patente en [IIT, 87]. La evaluación de soluciones se retrasaba considerablemente debido a un sólo proceso: el cálculo del número de ciclos térmicos en un satélite artificial en función de los parámetros de su órbita:

$N = P_c(R, i)$ si es una órbita circular

$N = P_e(i, a, b)$ si es una órbita elíptica

donde:

N	número de ciclos térmicos
R	radio de la órbita circular
i	inclinación del plano de la órbita
a, b	semiejes de la elipse

Las funciones P_c y P_e son en realidad procesos iterativos exageradamente lentos. Hay que comprobar si el satélite está iluminado o no por el Sol para varias posiciones de la tierra alrededor del Sol, para varios ciclos de la órbita y para muchas posiciones dentro de cada ciclo.

Se encontró otro ejemplo claro de aplicación de los ábacos lineales para proyectos ya terminados ([Cuadra et al, 85] y [García et al, 87]). Este es el cálculo de coeficientes de inducción propia y mutua de bobinas coaxiales:

$$L = P(a, l, t)$$

$$M = Q(a_i, a_e, l, t_i, t_e)$$

donde:

L	coeficiente de autoinducción

a	radio medio de la bobina
l	altura de la bobina
t	espesor del arrollamiento
M	coeficiente de inducción mutua
a_i, a_e	radios medios de las bobinas interior y exterior
t_i, t_e	espesores de las bobinas interior y exterior
l	altura (se consideran las dos bobinas con la misma altura)

Los procesos P y Q son desarrollos en serie tomados de [Bristol, 45]. Estos desarrollos son de convergencia lenta y con puntos de redefinición, es decir, la convergencia depende mucho del valor de las variables. Esto hace que sea arriesgado despreciar términos de los desarrollos, pues en algunos casos el error es importante.

El problema de la interpolación lineal en una o varias variables es que los puntos que se seleccionan como muestra tienen que ser cuidadosamente seleccionados. Se pretende conseguir la mayor precisión posible con el menor número de puntos almacenados, y esto exige una herramienta informática capaz de estudiar exhaustivamente las funciones que se desean sustituir por ábacos.

Con este fin se desarrolló e implantó el creador de ábacos lineales (CAL). El resultado es una herramienta informática que crea dinámicamente un ábaco de una función escalar de variable vectorial, incrementando el número de puntos muestra hasta obtener la precisión deseada. La dimensión de la variable vectorial puede ser cualquiera (CAL admite hasta 6 variables, que es un límite más que razonable para este tipo de ábacos); tanto las variables como la función pueden ser discretas o continuas, pues todas las variables se tratan como discretas pero con distintos tamaños.

El usuario de CAL puede seleccionar el tamaño de las variables independientes (incrementos y rangos), el número máximo de puntos por ábaco elemental (recuérdese que en zonas conflictivas de la función se crean ábacos auxiliares con un mallado más fino) y el margen de error máximo que está dispuesto a admitir entre la interpolación y la función original. Esta función original debe estar también programada en lenguaje C para incorporarse al proceso.

CAL no se limita a proporcionar los puntos muestra seleccionados, sino que **escribe el código** en lenguaje C del ábaco encargado de sustituir a la función. La herramienta está

programada en lenguaje C, y la desarrolló el autor de esta tesis en un período de 6 meses durante 1988. Esto fue posible gracias a la utilización sistemática de la metodología ANA de representación, pues la descomposición sistemática en subproblemas es útil tanto para un grupo de personas trabajando en paralelo como para una persona trabajando secuencialmente.

Una aplicación real de CAL se encuentra en el proyecto SEQA, descrito en [Sanz et al, 89]. En dicho proyecto se construyó un ábaco de interpolación para una tabla de doble entrada que expresaba la concentración de amoniaco en el agua en función del PH y la conductividad específica. La tabla se pasó a un ábaco lineal introduciendo manualmente los puntos, y posteriormente se aplicó CAL a dicho ábaco, reduciéndose en un 80% el número de puntos muestra y creando por tanto (automáticamente) un ábaco mucho más eficaz.

Otra aplicación real de CAL se encuentra en [Cuadra et al, 90], en donde se construyó una subrutina que llamaba a un ábaco lineal de cuatro ábacos distintos, según el valor de las variables de entraba. Los ábacos fueron construidos como en el ejemplo anterior a partir de tablas experimentales. La función que se interpolaba expresaba el aumento de reluctancia en el entrehierro por estar desalineados los dientes de rótor y estátor en un motor eléctrico.

Capítulo 8

Conclusiones, Aportaciones y Futuros Desarrollos

Los dos grandes temas estudiados en esta tesis se resumen en su título. Por un lado, el problema general de la optimización de diseño; por otro, la Ingeniería del Conocimiento, que hace posible resolver dicho problema por medio de ordenadores. Ambos temas se funden en la idea central y aportación principal de la tesis: la Optimización Estructurada Multiatributo.

El camino seguido para la realización de este trabajo parte de la experiencia en proyectos reales de optimización de diseño por ordenador, siendo la Optimización Estructurada Multiatributo el resultado de la generalización y formalización de los problemas y soluciones que componen esta experiencia.

En el desarrollo de la tesis se ha recorrido también el camino opuesto para llegar a la Optimización Estructurada Multiatributo. A partir de la literatura revisada (Lógica, Ingeniería del Conocimiento y Diseño) se ha modelado el problema de la optimización de diseño por ordenador, superándose para ello las etapas previas de estudio y formalización de dicho problema. En la exposición de las conclusiones fundamentales de la tesis se va a seguir el segundo camino por permitir una exposición más ordenada y racional.

8.1 - Resumen y Conclusiones.

Una actividad fundamental en la realización de esta tesis ha sido enmarcar adecuadamente el problema general de la optimización de diseño por ordenador en una disciplina tan amplia

como la Ingeniería del Conocimiento. En primer lugar se resumen las conclusiones obtenidas de esta labor (apartado 8.1.1) para lograr una comprensión más completa del planteamiento de la tesis. A continuación se resumen las conclusiones relativas al estudio del problema del diseño por ordenador dentro del marco establecido (apartado 8.1.2).

8.1.1 - Marco de la Tesis.

La **Ingeniería del Conocimiento** es la disciplina que intenta hacer realidad las aspiraciones de la tradición algorítmica (como corriente de pensamiento en Lógica) aplicando el desarrollo tecnológico. La Ingeniería del Conocimiento intenta aprovechar los conocimientos de Lógica para que las máquinas sean capaces de resolver problemas, o lo que es lo mismo, sean capaces de simular el pensamiento.

Cualquier sistema automático capaz de resolver un problema es un proyecto de Ingeniería del Conocimiento. La complejidad del problema y la sofisticación del método de resolución establecen diferencias cuantitativas pero no cualitativas entre los distintos proyectos. Los conocimientos de Lógica serán siempre necesarios, pues la Lógica estudia la relación entre conceptos y los criterios de validez y verdad de los razonamientos.

Esta actividad se considera una rama de la Ingeniería por dos razones: es la aplicación práctica de un conocimiento científico (Lógica, modelos del pensamiento) y trabaja con modelos matemáticos de dicho conocimiento científico. El trabajo en Ingeniería del Conocimiento es análogo al de las demás ingenierías: construir, aplicar y validar modelos matemáticos de realidades complejas, basándose en los conocimientos científicos y utilizando sistemas gráficos de representación para enfrentarse a la complejidad.

El proceso de desarrollo de un proyecto de Ingeniería del Conocimiento consta de tres etapas fundamentales:

1) La adquisición del conocimiento sobre el problema por resolver, que produce una descripción formal y un modelo conceptual del problema y de su método de resolución.

2) El modelado matemático exhaustivo del problema y de su resolución, que produce un algoritmo (aunque puede emplearse un modelo lingüístico intermedio). Un algoritmo se compone de una estructura de variables y otra estructura de funciones estrechamente relacionadas entre sí.

3) La implantación informática del algoritmo, que produce un programa de ordenador.

El método de trabajo consiste en aplicar al problema procesos sistemáticos de análisis y síntesis. Normalmente serán necesarias varias iteraciones a través de estas tres etapas, obteniéndose versiones cada vez mejores del modelo, del algoritmo y del programa. Será siempre muy útil contar con un sistema gráfico de representación del conocimiento, especialmente si este sistema es capaz de representar fielmente modelos, algoritmos y programas.

Un problema crítico es el de la comunicación del hombre con la máquina. Hay que traducir datos y pensamientos más o menos abstractos a un sistema compuesto de memorias y procesadores, o sea, a un ordenador. La utilización conjunta de un lenguaje gráfico (el más alto nivel posible, al menos en problemas de Ingeniería) y un lenguaje de programación de bajo nivel permite utilizar al máximo los recursos humanos y explotar eficazmente los recursos de la máquina.

Todo proceso automático de razonamiento es una forma de razonamiento funcional. Los conceptos de "información" (datos y soluciones) y "razonamiento" propios de la resolución de problemas, encajan perfectamente con los conceptos de "variable" y "función" comunes a la Lógica, las Matemáticas y la Informática, y también con la estructura física de las memorias y procesadores del ordenador.

Un proyecto de Ingeniería del Conocimiento puede considerarse un proceso de diseño de algoritmos y programas (software), estando estrechamente relacionado con el diseño de circuitos lógicos (hardware). La diferencia fundamental está en el nivel de abstracción teóricamente ilimitado que puede alcanzarse con el software.

8.1.2 - La Optimización Estructurada Multiatributo.

En esta tesis se aplica la Ingeniería del Conocimiento al **problema general de diseño en Ingeniería** siguiendo el proceso descrito. La etapa de adquisición del conocimiento da como resultado una formalización del problema y un modelo conceptual del método de resolución del mismo. Se escoge un modelo matemático (en vez de un modelo lingüístico) por ser más adecuado al diseño en Ingeniería, con excepción de algunas aplicaciones especiales. En

cualquier caso los modelos lingüísticos siempre tendrán que traducirse a modelos matemáticos al desarrollarse el algoritmo de resolución del problema.

La **Optimización Estructurada Multiatributo** resulta de aplicar sistemáticamente procesos de análisis y síntesis al problema de optimización de diseño. El resultado es una estructura de subproblemas relacionados entre sí; para algunos subproblemas se proponen soluciones ya probadas, mientras que para otros sólo se pueden sugerir directrices y posibles líneas de trabajo.

El proceso general de diseño se descompone en un conjunto de **fases** resueltas secuencialmente:

- Una fase inicial interactiva llamada **análisis de consistencia y prediseño.**

- Una fase final que informa sobre la resolución del proceso llamada **interfaz final.**

- Una serie de **fases de diseño,** utilizando cada fase un modelo del sistema más detallado que la fase anterior y dejando el problema más definido para la fase siguiente.

El proceso de **análisis de consistencia y prediseño** se puede descomponer en varios subproblemas: garantizar que los requisitos del problema sean coherentes y completos (corrección formal del problema), encontrar soluciones aproximadas que sirvan de puntos iniciales a la primera fase de diseño y reducir lo más posible la dimensión y el tamaño de las regiones de búsqueda.

Cada fase de diseño es una **optimización multiatributo,** y su misión es encontrar un conjunto de **puntos de interés** (seleccionados con criterios multiatributo) a partir de un conjunto de **puntos iniciales** seleccionados por la fase anterior. Cada fase de diseño se descompone a su vez en un **conjunto estructurado de subproblemas:**

- Problemas de **optimización,** llamados **optimizaciones parciales,** organizados como optimizaciones anidadas y optimizaciones paralelas.

- Problemas de **convergencia** entre las optimizaciones paralelas.

No hay un criterio general para resolver los problemas de convergencia de optimizaciones paralelas. La única forma de abordarlos es aplicar el conocimiento específico del problema de

diseño planteado. La **jerarquización** de las **optimizaciones parciales** (o lo que es lo mismo, de las variables independientes) se puede basar en criterios generales, como son el tamaño de las variables y su comportamiento local, pero siempre es conveniente que se complementen estos criterios con otros específicos del problema.

Cada **optimización parcial** es un problema de optimización completo, que podrá ser multiatributo o monoatributo; lo interesante es que si los criterios de descomposición en optimizaciones son adecuados, el problema presentará características matemáticas deseables (dimensión constante, variables independientes homogéneas, región factible conexa, etc.) y además el conocimiento específico del problema podrá aplicarse fácilmente para agilizar la búsqueda.

Una optimización parcial se descompone a su vez en los **subproblemas** de evaluación, estrategia de búsqueda y selección multiatributo:

- La **evaluación** puede plantear dificultades por falta de convergencia y/o lentitud en los procesos de cálculo.

- La **estrategia de búsqueda** dependerá fundamentalmente del tamaño y el comportamiento local de las variables implicadas.

- La **selección multiatributo** se realizará según sugiera la naturaleza del problema, apareciendo como dificultad especial el tratamiento adecuado de la incertidumbre.

Una vez descompuesto el problema general de la optimización de diseño cada subproblema podrá ser abordado mediante una técnica distinta, descentralizándose la toma de decisiones y pudiéndose aplicar así con más eficacia el conocimiento específico sobre la naturaleza del problema.

El conocimiento específico de cada problema de diseño se aplica en dos modos muy diferentes:

- Durante la fase de desarrollo de la herramienta, para seleccionar las variables independientes, los atributos, niveles de detalle en el modelado, fases de diseño, jerarquías de las variables, etc.

- Para incorporarlo en los procesos de decisión de la herramienta, especialmente en las estrategias de búsqueda y los controles de convergencia. Este tipo de conocimiento puede incrementarse automáticamente si se implantan sistemas de aprendizaje en los distintos subproblemas.

El nivel de abstracción del modelo de diseño propuesto permite dotar a la herramienta de recursos que simulen un comportamiento creativo: del mismo modo que se evalúa un diseño se puede evaluar la eficacia de una estrategia, la precisión de un modelo (márgenes de incertidumbre) o un criterio dado de selección multiatributo.

8.2 - Aportaciones de la Tesis.

En cuanto a las aportaciones principales de la tesis, la más importante es el desarrollo de la metodología general de modelado del problema general de diseño y su descomposición en subproblemas, es decir, la Optimización Estructurada Multiatributo. La segunda en importancia es la visión integral de la Ingeniería del Conocimiento y de los proyectos englobados en esta disciplina. La tercera es un conjunto de técnicas de apoyo del problema del diseño que pueden también aplicarse en contextos muy diferentes, como por ejemplo la creación de ábacos de interpolación lineal y el método de representación ANA.

A continuación se citan las aportaciones originales de la tesis agrupadas según los tres bloques citados. Debe aclararse que se usa el término "aportación original" para las ideas que tienen su origen en el desarrollo de la tesis (o en los proyectos que la sustentaron) y además no coincidan con ideas anteriores según la literatura técnica revisada.

8.2.1 - La Optimización Estructurada Multiatributo.

En lo que respecta a aspectos concretos relacionados con la Optimización Estructurada Multiatributo pueden citarse las siguientes aportaciones originales:

- La elección de una nomenclatura completa y sin ambigüedades para el problema de optimización de diseño. Algunos términos son además originales (o al menos lo es el

uso que se hace de ellos), como por ejemplo los requisitos, el espacio de restricciones y la región de búsqueda.

- La definición de óptimo local en función de la estrategia de búsqueda seguida y del tamaño de los incrementos mínimos de las variables independientes.

- El orden sistemático propuesto para adquirir el conocimiento relativo a un problema de optimización de diseño.

- El hecho de considerar todas las variables como discretas, y la definición de los criterios de "tamaño" y "comportamiento local" como criterios prácticos para establecer jerarquías entre las variables independientes y para seleccionar la estrategia de búsqueda más apropiada.

- El algoritmo propuesto para resolver un problema genérico de optimización de diseño, es decir, la descomposición detallada del problema en fases de diseño, optimizaciones parciales, etc., con sus intercambios de datos y secuencias de actuación.

- La resolución de búsquedas en espacios de dimensión variable mediante optimizaciones parciales anidadas.

- La formalización matemática de la descomposición de un problema en optimizaciones parciales, así como la definición de los objetivos y la interpretación geométrica de tal descomposición.

- La distinción entre búsquedas en árboles de soluciones completas y búsquedas en árboles de soluciones incompletas.

- La discusión sobre el planteamiento y resolución de un problema de etiquetado consistente como un problema de optimización con variables continuas y búsquedas locales.

- La búsqueda en espacio heurístico para abordar problemas con un gran número de variables.

- La definición de hipersuperficie óptima como el resultado teórico de una optimización multiatributo.

- El criterio de selección multiatributo por optimalidad respecto a un conjunto de funciones lineales de los atributos, y la idea asociada de aproximar la hipersuperficie óptima por hiperplanos tangentes hallados mediante búsquedas locales.

- La aplicación de la selección multiatributo a las búsquedas en árbol ayudadas por heurísticos.

- La selección multiatributo de alternativas de alto nivel por comparación entre las hipersuperficies óptimas asociadas a las mismas en niveles inferiores.

8.2.2 - La Ingeniería del Conocimiento.

Se pueden citar las siguientes aportaciones relacionadas con la Ingeniería del Conocimiento:

- Una definición de Ingeniería del Conocimiento y su caracterización como una disciplina específica que integra la Lógica, la Ingeniería y la Informática; en particular, la idea de considerar a la Ingeniería del Conocimiento como una continuación de la tradición algorítmica y la importancia concedida al concepto de "algoritmo".

- La asociación de los conceptos de clases, individuos y propiedades (Lógica) con espacios vectoriales, puntos y componentes de un vector (Matemáticas) y las memorias y procesadores de la máquina como argumento para elegir un modelo procedural.

- La formalización de la resolución de un problema como una función definida entre dos conjuntos (o espacios vectoriales asociados), el de datos y el de soluciones, y la del algoritmo de resolución del problema como la ley de obtención de imágenes de dicha función.

- La descripción de un algoritmo como la integración de dos estructuras complejas estrechamente relacionadas, una de variables y otra de funciones, y la enumeración sistemática de las relaciones dinámicas y estáticas entre variables y funciones.

- La estructura de "red de árboles", tanto el término como su definición.

- Un proceso sistemático de desarrollo de proyectos de Ingeniería del Conocimiento y su descripción detallada.

- La complementariedad de los lenguajes gráficos de programación (como ANA) y los lenguajes de bajo nivel, y en particular la discusión sobre el lenguaje C.

- La discusión sobre flexibilidad y rigidez de la arquitectura informática y su relación con la creatividad y la innovación en las soluciones propuestas por una herramienta.

8.2.3 - Técnicas y Desarrollos Auxiliares.

Las técnicas desarrolladas como apoyo tanto del problema de diseño por ordenador como de la Ingeniería del Conocimiento en general son:

- El método de representación ANA y la herramienta informática capaz de generar interactivamente los diagramas ANA.

- La expresión analítica recursiva de la interpolación lineal en N dimensiones.

- El algoritmo y la herramienta automática de creación de ábacos lineales en N dimensiones (CAL).

- Las memorias selectivas y su aplicación a la creación de ábacos no lineales en espacios de un número muy elevado de dimensiones.

8.3 - Futuros Desarrollos.

Los **futuros desarrollos** están básicamente centrados en la fase de análisis de consistencia y prediseño, que por otro lado es la más difícil de generalizar por depender mucho de la naturaleza de cada problema específico de diseño. Es por ello la más adecuada para la aplicación de las técnicas clásicas de la Ingeniería del Conocimiento, como puedan ser los sistemas de reglas de producción, el razonamiento cualitativo, la analogía (reconocimiento de patrones), etc.

Merece especial atención la incorporación de sistemas de aprendizaje automático (memorias asociativas, memorias selectivas, creación automática de reglas, etc.) para la selección de diseños iniciales y regiones de búsqueda en función de los requisitos y para resolver problemas de convergencia de las optimizaciones parciales.

También es interesante la automatización del tratamiento de normativas (estándares) para generar regiones de búsqueda y sistemas compatibles de restricciones. Estos problemas se pueden tratar según los casos como propagación de restricciones, etiquetado consistente o análisis de monotonía.

Quedan por aplicar a casos reales ideas como la búsqueda en espacio heurístico, los ábacos no lineales y las memorias selectivas. Pero quizá el trabajo más importante que quede para el futuro sea la resolución de un caso de aplicación industrial con la complejidad y el interés suficientes (interés tanto práctico como teórico) como para mostrar concluyentemente la eficacia de la Optimización Estructurada Multiatributo.

BIBLIOGRAFIA

A

[Adiba & Nguyen, 85]
M. Adiba, G.T. Nguyen, **"Knowledge Engineering for CAD/VLSI on a Generalized Data Management System."** "Knowledge Engineering in CAD", J.S. Gero ed., North Holland, 1985.

[Agogino & Almgren, 87]
Alice M. Agogino, Ann S. Almgren, **"Symbolic Computation in Computer-Aided Optimal Design."** "Expert Systems in CAD", J.S. Gero ed., North Holland 1987.

[Allen, 90]
Jonathan Allen, **"Performance-Directed Synthesis of VLSI Systems."** Proceedings of the IEEE, Vol. 78 NO. 2, February 1990.

[Appelbaum et al, 87]
J. Appelbaum, E.F. Fuchs, J.C. White, **"Optimization of Three-Phase Induction Motor Design."** IEEE Transactions on Energy Conversion, Vol. EC-2, No. 3, September 1987.

[Arbab, 89]
Farhad Arbab, **"Design Object Modeling."** "Intelligent CAD I", H. Yoshikawa, D. Gossard eds., North Holland 1989.

B

[Baker, 89]
Louis Baker, **"C Tools for Scientists and Engineers."** McGraw-Hill Inc., 1989.

[Balas & Martin, 80]
Egon Balas, Clarence H. Martin, **"Pivot and Complement- A Heuristic for 0-1 Programming."** Management Science, Vol 26 No 1, January 1980.

[Barbuceanu, 85]
Mihai Barbuceanu, **"An Object-Centered Framework for Expert Systems in Computer-Aided Design."** "Knowledge Engineering in CAD", J.S. Gero ed., North Holland, 1985.

[Barr & Feigenbaum, 81]
"The Handbook of Artificial Intelligence." Avrom Barr, Edward A. Feigenbaum, (ed.), William Kaufman Inc., Los Altos, California, 1981.

[Beale, 77]
E.M.L. Beale, **"Integer Programming."** "The State of the Art in Numerical Analysis, pp. 409-448, D. Jacobs ed., Academic Press, London 1977.

[Benders, 62]
J.F. Benders, **"Partitional Procedures for Solving Mixed-Variables Programming Problems."** Numerische Mathematik 4, pp. 238-258, 1962.

[Bird & Wadler, 88]
Richard Bird, Philip Wadler, **"Introduction to Functional Programming."** Prentice Hall International (UK) Ltd., 1988.

[Birewar, 90]
Deepak D. Birewar, **"Design, Planning and Scheduling of Multiproduct Batch Plants."** Ph.D. Thesis, Carnegie Mellon University, Pittsburg (Pennsylvania), EDRC 02-11-90.

[Borisov & Naglis, 85]
Arkady Borisov, Leonards Naglis, **"Multi-Criteria Choice of Alternatives in an Expert System for Computer-Aided Design of Industrial Robot Installations."** Fuzzy Sets and Systems, No. 16 pp.93-101, North Holland 1985.

[Borrel, 87]
J.M. Borrel Fontelle, **"Métodos Matemáticos para la Economía. Tomo 2: Programación Matemática."** Editorial Pirámide, Madrid 1987.

[Bowen, 85]
J.A. Bowen **"Automated Configuration of Hardware for Dedicated Microprocessor Application Systems."** "Knowledge Engineering in CAD", J.S. Gero ed., North Holland, 1985.

[Brayton et al, 90]
R.K. Brayton, G.D. Hatchel, A.L. Sangiovanni-Vincentelli, **"Multilevel Logic Synthesis."** Proceedings of the IEEE, Vol. 78 NO. 2, February 1990.

[Bristol, 45]
Herbert Bristol Dwight, **"Electrical Coils and Conductors."** McGraw-Hill, 1945.

[Brown & Chandrasekaran, 85]
D.C. Brown, B. Chandrasekaran **"Expert Systems for a Class of Mechanical Design Activity."** "Knowledge Engineering in CAD", J.S. Gero ed., North Holland, 1985.

C

[Carbonell, 85]
J.G. Carbonell, **"Derivational Analogy: a Theory of Reconstructive Problem Solving and Expertise Adquisition."** Tech. Report CMU-CS-85-115, Dep. of Computer Science, Carnegie Mellon University, March 1985.

[Carnap, 47]
R. Carnap, **"Meaning and Neccesity (A Study in Semantics and Modal Logic)."** Chicago y Londres, The University of Chicago Press, 1947. 2ª edición ampliada, 1956.

[Chankong & Haimes, 83]
V. Chankong, Y.Y. Haimes, **"Optimization-Based Methods for Multiobjective Decision-Making: An Overview."** Large Scale Systems: Theory and Applications, Vol. 5, No. 1, August 1983.

[Charniak & McDermott, 85]
Eugene Charniak, Drew McDermott, **"Artificial Intelligence."** Addison-Wesley Publishing Company Inc., 1985.

[Cholvy & Foisseau, 85]
Laurence Cholvy, Jack Foisseau **"Encoding Requirements to Increase Modelisation Assistance."** "Knowledge Engineering in CAD", J.S. Gero ed., North Holland, 1985.

[Cloarec et al, 85]
J.F. Cloarec, J.F. Cudelou, J. Collet **"A Configurer for Switching System Software Based on the Knowledge about the Assembling of Program Modules."** "Knowledge Engineering in CAD", J.S. Gero ed., North Holland, 1985.

[Cook, 86]
Steve Cook, **"Languajes and Object-Oriented Programming."** Software Engineering Journal, Marzo 1986.

[Coyne et al, 87]
Richard D. Coyne, Michael A. Rosenman, Anthony D. Radford, John S. Gero, **"Innovation and Creativity in Knowledge-Based CAD."** "Expert Systems in CAD", J.S. Gero ed., North Holland 1987.

[Cuadra et al, 85]
F. Cuadra, F.J. Pérez Thoden, J.I. Pérez Arriaga, **"Optimización con Ayuda de Ordenador del Diseño de Reactancias."** ICAI, Universidad Pontificia Comillas, 1985.

[Cuadra, 90]
F. Cuadra, **"CAE Techniques: Application to Dynamic Cargo Accommodation / Location."** ESTEC Contract No.8607/89/NL/MAC, Technical Note 01, February 1990.

[Cuadra et al, 90]
F. Cuadra, J.I. Pérez Arriaga, J.H. Lang, **"The Application of Knowledge Engineering to the Computer-Aided Design of Optimal Electric Drives."**

International Conference on Electrical Machines, Cambridge (Massachusetts) USA, 12-15 August 1990.

[Cuadra & León, 90]
F. Cuadra, J.J. León, "**CAE Techniques: Application to Dynamic Cargo Accommodation / Location.**" ESTEC Contract No. 8607/89/NL/MAC, Final Report, April 1990.

D

[Dahl et al, 72]
O. Dahl, E.W. Dijkstra, C.A.R. Hoare, "**Structured Programming.**" Academic Press, 1972.

[De Kleer & Sussman, 80]
Johan de Kleer, Gerald Jay Sussman, "**Propagation of Costraints Applied to Circuit Synthesis.**" Circuit Theory and Applications, Vol. 8 pp. 127-144, 1980.

[De Man et al, 90]
Hugo de Man, Francky Catthoor, Gert Goosens, Jan Vanhoof, Jef Van Meerbergen, Stefaan Note, Jos Huisken, "**Architecture-Driven Synthesis Techniques for VLSI Implementation of DSP Algorithms.**" Proceedings of the IEEE, Vol. 78 NO. 2, February 1990.

[Deaño, 83]
Alfredo Deaño, "**Introducción a la Lógica Formal.**" Alianza Editorial, 1983 (primera edición en 1974).

[Dietterich & Ullman, 87]
T.G. Dietterich, D.G. Ullman, "**FORLOG: A Logic-Based Architecture for Design.**" "Expert Systems in CAD", J.S. Gero ed., North Holland 1987.

[Dorbyshire & Davies, 85]
I. Dorbyshire, B.J. Davies, "**EXCAP: An Expert Generative Process Planning System.**" "Knowledge Engineering in CAD", J.S. Gero ed., North Holland, 1985.

[Duran & Grossman, 86]
M.A. Duran, I.E. Grossman, "**An Outer-Approximation Algorithm for a Class of Mixed-Integer Non-Linear Programs.**" Mathematical Programming 36, pp. 307-339, 1986.

E

[Eberling & Wu, 89]
C. Eberlig, Z. Wu, "**Wirelisp: Combining Graphics and Procedures in a Circuit Specification Language.**" Proceedings IEEE Int. Conf. CAD, Carnegie Mellon University, July 1989.

[Elias, 85]
Antonio L. Elias, "**Knowledge Engineering of the Aircraft Design Process**" in Kowalik, J.S.(ed.), Knowledge Based Problem Solving, Englewood Cliffs, N.J. Prentice-Hall Chapter 6., 1985.

[Etherington, 88]
David W. Etherington, "**Reasoning with Incomplete Information.**" Pitman Publishing, London 1988.

F

[Fei et al, 89]
R. Fei, E.F. Fuchs, H. Huang, "**Comparison of Two Optimization Techniques as Applied to Three-Phase Induction Motor Design.**" IEEE Transactions on Energy Conversion, Vol. 4, No. 4, December 1989.

[Fenves & Baker, 87]
Steven J. Fenves, Nelson C. Baker, "**Spatial and Functional Representation Languaje for Structural Design.**" "Expert Systems in CAD", J.S. Gero ed., North Holland 1987.

[Field & Harrison, 88]
Anthony J. Field, Peter G. Harrison, "**Functional Programming.**" Addison-Wesley Publishing Company Inc., 1988.

[Findler, 79]
"**Associative Networks: the Representation and Use of Knowledge by Computers.**" N.V. Findler (ed.). Academic Press, New York 1979.

[Freeman & Newell, 71]
P. Freeman, A. Newell, "**A Model for Functional Reasoning in Design**" Proceedings IJCAI pp. 621-640, 1971.

[Freuder, 76]
Eugene C. Freuder, "**Synthesizing Constraint Expressions.**" Artificial Intelligence Laboratory, MIT 1976.

G

[Galiana & McGillis, 88]
F.D. Galiana, D. McGillis, "**Design of a Longitudinal EHV Transmission Network: A Knowledge-Based Approach.**" Symposium of Expert Systems Application to Power Systems, Stockholm-Helsinki, 1988.

[García et al, 87]
E. García, F. Cuadra, J.I. Pérez Arriaga, "**Diseño Asistido por Ordenador de Bobinas de Bloqueo.**" ICAI, Universidad Pontificia Comillas, 1987.

[Gardies, 75]
J.L. Gardies, "**La Logique du Temps.**" París, PUF.,1975.

[Gero et al, 85]
J.S. Gero, A.D. Radford, R. Coyne, V.T. Akiner, "**Knowledge-Based Computer-Aided Architectural Design.**" "Knowledge Engineering in CAD", J.S. Gero ed., North Holland, 1985.

[Gill et al, 81]
P.E. Gill, W. Murray, M.H. Wright, "**Practical Optimization.**" Academic Press Inc., London, 1981.

[Green & Brown, 87]
Douglas S. Green, David C. Brown, "**Qualitative Reasoning during Design about Shape and Fit.**" "Expert Systems in CAD", J.S. Gero ed., North Holland 1987.

[Grossman, 90]
Ignacio Grossman, "**Mixed-Integer Nonlinear Programming Techniques for the Synthesis of Engineering Systems.**" Carnegie Mellon University, EDRC 06-83-90.

[Gupta, 88]
Anurag P. Gupta, "**A Hierarchical Problem-Solving Architecture for Design Synthesis of Single Board Computers.**" EDRC-18-04-88, Carnegie Mellon University, Pittsburgh, Pennsylvania, 1988.

H

[Han et al, 87]
G. Han, S. Ohsuga, H. Yamauchi, "**The Application of Knowledge Base Technology to CAD.**" "Expert Systems in CAD", J.S. Gero ed., North Holland 1987.

[Haralick & Saphiro, 79]
R.M. Haralick, L. Shapiro, "**The Consistent Labelling Problem: Part 1.**" IEEE Transactions Pattern Analysis and Machine Intelligence, Vol PAMI-1, p.173, 1979.

[Harrah, 63]
D. Harrah, "**Communication: A Logical Model.**" Cambridge, Mass., The MIT Press, 1963.

[Harrison et al, 90]
David S. Harrison, A. Richard Newton, Rick L. Spickelmier, Timothy J. Barnes, "**Electronic CAD Frameworks.**" Proceedings of the IEEE, Vol. 78 NO. 2, February 1990.

[Hatvany, 85]
J. Hatvany, **"The Missing Tools of CAD for Mechanical Engineering."** "Knowledge Engineering in CAD", J.S. Gero ed., North Holland, 1985.

[Hayes-Roth, 85]
B. Hayes-Roth, **"A Blackboard Architecture for Control."** Artificial Intelligence 26, 1985.

[Hooke & Jeeves, 61]
R. Hooke, T.A. Jeeves, **"Direct Search: Solution of Numerical and Statistical Problems."** Journal of the A.C.M., pp 212-229, 1961.

[Hummel & Zucker, 83]
R.A. Hummel, S.W. Zucker, **"On the Foundations of Relaxation Labelling Processes."** IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. PAMI-5, No. 3, 1983.

I

[IIT, 87]
Instituto de Investigación Tecnológica. **"Study on Conceptual Design of Spacecrafts Using Computer-Aided Engineering Techniques."** ESTEC contract 6886/85/NL/PP Final Report, December 1987.

[Ishii & Barkan, 87]
Kasuki Ishii, Philip Barkan, **"Rule-Based Sensitivity Analysis: a Framework for Expert Systems in Mechanical System Design."** "Expert Systems in CAD", J.S. Gero ed., North Holland 1987.

[Iwata et al, 87]
Kazuaki Iwata, Nobuhiro Sugimura, Woonkyu Lee, **"Knowledge-Based Recognition of Hand-Written Drawings for Product Modelling in Machine Design."** "Expert Systems in CAD", J.S. Gero ed., North Holland 1987.

J

[James et al, 85]
John R. James, Piero P. Bonissone, Dean K. Frederick, James H. Taylor, **"A Retrospective View of CACE-III: Considerations in Coordinating Symbolic and Numeric Computation in a Rule-Based Expert System."** IEEE CH2215-2/85/0000/0532$01.00, 1985.

[Jazdzynski, 89]
W. Jazdzynski, **"Multicriterial Optimisation of Squirrel-Cage Induction Motor Design."** IEEE Proceedings, Vol. 136, Pt. B, No. 6, November 1989.

K

[Kalay et al, 87]
Yehuda E. Kalay, Lucien M. Swerdloff, Anton C. Harfmann, **"A Knowledge-Based Computable Model of Design."** "Expert Systems in CAD", J.S. Gero ed., North Holland 1987.

[Kamal et al, 87]
Saiyid Z. Kamal, H.M. Karandikar, Farrokh Mistree, Douglas Muster, **"Knowledge Representation for Discipline Independent Decision Making."** "Expert Systems in CAD", J.S. Gero ed., North Holland 1987.

[Kermanshahi et al, 90]
Bahman S. Kermanshahi, Y. Wu, Keiichiro Yasuda, Ryuichi Yokoyama, **"Environmental Marginal Cost Evaluation by Non Inferiority Surface."** IEEE/Pes 1990 Winter Meeting, Atlanta (Georgia), February 1990.

[Kernighan & Ritchie, 88]
Brian W. Kernighan, Dennis M. Ritchie, **"The C Programming Languaje."** Prentice Hall, Software Series, 1988.

[Kolb, 86a]
Mark A. Kolb, "**On the Numerical Solution of Simultaneous Non-Linear Equations in Computer-Aided Preliminary Design.**" S.M.Thesis, Department of Aeronautics and Astronautics, MIT January 1986.

[Kolb, 86b]
Mark A. Kolb, "**Rubber Airplane: a Paradigm for Computer-Aided Preliminary Design Based on Component Modelling.**" Doctoral thesis proposal, Department of Aeronautics and Astronautics, MIT March 1986.

[Kramer, 87]
Glenn A. Kramer, "**Incorporating Mathematical Knowledge into Design Models.**" "Expert Systems in CAD", J.S. Gero ed., North Holland 1987.

[Krishnan et al, 88]
R. Krishnan, Aravind S. Bharadwaj, Peter N. Materu, "**Computer-Aided Design of Electrical Machines for Variable Speed Applications.**" IEEE Transactions on Industrial Electronics, Vol. 35, No. 4, November 1988.

[Kuh & Ohtsuki, 90]
Ernest S. Kuh, Tatsuo Ohtsuki, "**Recent Advances in VLSI Layout.**" Proceedings of the IEEE, Vol. 78 NO. 2, February 1990.

[Kuratowsky, 66]
Kamiriez Kuratowsky, "**Introducción a la Teoría de Conjuntos y a la Topología.**" Vicens Vives, 1966.

L

[Lajoie, 86]
Ronnie M. Lajoie, "**Paper Airplane Research. Multiple Valued Output and External Code Interface Capability.**" Final Report, Flight Transportation Laboratory, MIT December 1986.

[Larousse, 67]
"**Gran Enciclopedia Larousse.**" Edición española en diez volúmenes, Editorial Planeta S.A., 1967.

[Latorre et al, 90]
G. Latorre, J.I. Pérez Arriaga, A. Ramos, J. Román, J.F. Alonso, A. Sáiz, "**Un Modelo de Planificación Estático de la Red de Transporte a Largo Plazo.**" Primeras Jornadas Hispano-Lusas de Ingeniería Eléctrica, Vigo, Julio 1990.

M

[Mac an Airchinnig, 85]
Micheal Mac an Airchinnig, "**CAD, KE and ADA.**" "Knowledge Engineering in CAD", J.S. Gero ed., North Holland, 1985.

[Maher & Fenves, 85]
M.L. Maher, S.J. Fenves. "**HI-RISE: An Expert System for the Preliminary Structural Design of High-Rise Buildings.**" "Knowledge Engineering in CAD", J.S. Gero ed., North Holland, 1985.

[Maher & Zhao, 87]
Mary Lou Maher, F. Zhao, "**Using Experience to Plan the Synthesis of New Designs.**" "Expert Systems in CAD", J.S. Gero ed., North Holland 1987.

[Mally, 90]
Wojciech Mally, "**Computer-Aided Design for VLSI Circuit Manufacturability.**" Proceedings of the IEEE, Vol. 78 NO. 2, February 1990.

[MCC, 89]
Microelectronics and Computer Corporation, "**C Module Editor: Version 2.0, Users Guide.**" Austin TX., 1989.

[McFarland et al, 90]
Michael C. McFarland, Alice C. Parker, Raul Camposano, "**The High-Level Synthesis of Digital Systems.**" Proceedings of the IEEE, Vol. 78 NO. 2, February 1990.

[Minsky, 75]
M. Minsky, "**A Framework for Representing Knowledge.**" "The Psycology of Computer Vision", pp. 211-277, P. Winston (ed.), McGraw-Hill, New York 1975.

[Mozumder & Strojwas, 90]
P.K. Mozumder, A.J. Strojwas, "**Statistical Control of VLSI Fabrication Processes.**" Proceedings of the IEEE, Vol. 78 NO. 2, February 1990.

[Murthy & Addanki, 87]
Seshashayee S. Murthy, Sanjaya Addanki, "**PROMPT: An Innovative Design Tool.**" "Expert Systems in CAD", J.S. Gero ed., North Holland 1987.

N

[Nanda et al, 87]
J. Nanda, D.P. Kothari, K.S. Lingamurthy, "**Economic Emission Load Dispatch through Goal Programming Techniques.**" Proceedings of the IEEE Pes Winter Meeting, USA 1987.

[Navinchandra & Marks, 87]
D. Navinchandra, D.H. Marks, "**Design Exploration through Constraint Relaxation.**" "Expert Systems in CAD", J.S. Gero ed., North Holland 1987.

O

[Ohsuga, 85]
Setsuo Ohsuga, "**Conceptual Design of CAD Systems Involving Knowledge Bases.**" "Knowledge Engineering in CAD", J.S. Gero ed., North Holland, 1985.

P

[Papalambros & Wilde, 79]
P. Papalambros, D.J. Wilde, "**Global Non-Iterative Design Optimization using Monotonocity Analysis.**" ASME Journal of Mechanical Design 101(3), pp. 645-649, October 1979.

[Pérez Arriaga, 78]
J.I. Pérez Arriaga, "**Computer Aided Design of High Speed Synchronous Machines.**" Massachusetts Institute of Technology, M.S. Thesis, Dept. of Electrical Engineering 1978.

Q

R

[Rainman, 86]
Oliver Rainman, "**Order of Magnitude Reasoning.**" Proceedings of the Fifth National Conference on Artificial Intelligence, 1 July 1986.

[Reiter, 80]
R. Reiter, "**A Logic for Default Reasoning.**" Artificial Intelligence 13, pp. 81-132, North-Holland Publishing Company, 1980.

[Rehak et al, 85]
D.R. Rehak, H. Craig Howard, Duvvuru Sriram, "**Requirements and Priciples for Intelligent CAD Systems.**" "Knowledge Engineering in CAD", J.S. Gero ed., North Holland, 1985.

[Rich, 83]
Elaine Rich, "**Artificial Intelligence.**" McGraw-Hill, Inc., 1983.

[Rodríguez-Piñero, 86]
Jorge Rodríguez-Piñero Fernández, "**Cálculo Infinitesimal.**" Instituto Católico de Artes e Industrias (ICAI), Universidad Pontificia Comillas, Madrid 1986.

S

[Sacristán, 64]
Manuel Sacristán, "**Introducción a la Lógica y al Análisis Formal.**" Ediciones Ariel, S.A., Barcelona 1964.

[San Gil, 90]
Juan José San Gil, "**Modelos de Representación del Conocimiento para Tratamiento de Normativas Industriales. Aplicación a la Limitación de Perturbaciones en Redes Eléctricas de Distribución.**" Tesis Doctoral, Universidad Pontificia Comillas, Madrid, 1990.

[Sanz et al, 89]
M.A. Sanz Bobi, J.I. Pérez Arriaga, J.L. Serrano Carballo, M.E. Ortiz Alfaro, J.J. Alba, A. Doménech, M. J. Villamediana, J. González Huerta, J.J. Fernández Martínez, "**Control and Diagnosis of Water Chemistry in the Water-Steam Cycle and Water Makeup in a Fossil Fueled Power Plant.**" Second Symposium on Expert Systems Application to Power Systems, Seattle (Washington) July 1989.

[Samuelson & Nordhaus, 85]
Paul A. Samuelson, William D. Nordhaus, "**Economics.**" Twelfth Edition, McGraw-Hill Inc., 1985.

[Schildt, 86]
Herbert Schildt, "**Advanced C.**" Osborne McGraw-Hill, Berkeley, California 1986.

[Schildt, 87]
Herbert Schildt, "**Artificial Intelligence Using C.**" Osborne McGraw-Hill, Berkeley, California 1987.

[Schweppe & Merril, 86]
F.C. Schweppe, H.M. Merril, "**Multiple Attribute Trade-Off Analysis.**" EPRI RP 2537, April 1986.

[Shah & Rogers, 88]
Jami J. Shah, Mary T. Rogers, "**Functional Requirements and Conceptual Design of the Feature-Based Modelling System.**" Computer-Aided Engineering Journal, February 1988.

[Shank & Abelson, 77]
R.C. Schank, R.P. Abelson, "**Scripts, Plans, Goals and Understanding.**" Hillsdale, N.J.: Lawrence Erlbaum, 1977.

[Shukla, 88]
S. Shukla, "**EDESYN: A Knowledge-Based Expert System Shell for Preliminary Engineering Design.**" EDRC-01-06-88, Carnegie Mellon University, Pittsburgh, Pennsylvania, 1988.

[Singh et al, 83]
Bhim Singh, B.P. Singh, S.S. Murthy, C.S. Jha, "**Experience in Design Optimization of Induction Motor using 'SUMT' Algorithm.**" IEEE Transactions on PAS, Vol. PAS-102, No. 10, October 1983.

[Stebbing, 65]
L. Susan Stebbing, "**Introducción a la Lógica Moderna.**" Fondo de Cultura Económica, Méjico 1965. Edición original "A Modern Elementary Logic" por Methuen & Co. Ltd., Londres 1943.

[Struss, 87]
Peter Struss, "**Multiple Representation of Structure and Function.**" "Expert Systems in CAD", J.S. Gero ed., North Holland 1987.

[Sutherland, 63]
I.E. Sutherland, "**Sketchpad : A Man-Machine Graphical Communication System.**" Proceedings of Spring Joint Computer Conference, 1963.

[Systems Designers, 85]
Systems Designers, "**CORE Manual (Controlled Requirements Expression).**" Camberley (Surrey UK), 1985.

[Szalapaj & Bijl, 85]
P.J. Szalapaj, A. Bijl, "**Knowing Where to Draw the Line.**" "Knowledge Engineering in CAD", J.S. Gero ed., North Holland, 1985.

T

[Tomiyama & Ten Hagen, 87]
Tetsuo Tomiyama, Paul J.W. Ten Hagen, **"Organization of Design Knowledge in an Intelligent CAD Environment."** "Expert Systems in CAD", J.S. Gero ed., North Holland 1987.

[Tomiyama & Yoshikawa, 85]
Tetsuo Tomiyama, Hiroyuki Yoshikawa, **"Requirements and Priciples for Intelligent CAD Systems."** "Knowledge Engineering in CAD", J.S. Gero ed., North Holland, 1985.

[Tommelein et al, 87]
Iris D. Tommelein, M. Vaughan Johnson Jr., Barbara Hayes-Roth, Raymond E. Levitt, **"SIGHTPLAN: A Blackboard Expert System for Construction Site Layout."** "Expert Systems in CAD", J.S. Gero ed., North Holland 1987.

U

V

[Van Fraasen, 69]
B.C. Van Fraasen, **"Presuppositions, Supervaluations and Free Logic"**, K.Lambert (ed.), "The.Logical Way of Doing Things." New Haven y Londres, Yale University Press, pp. 67 y ss., 1969.

[Veinott, 72]
Cyril G. Veinott, **"Computer-Aided Design of Electric Machinery"**, MIT Press, Cambridge (MA), 1972.

[Vemuri, 88]
"Artificial Neural Networks: Theoretical Concepts", V.Vemuri (ed.), "Neural Networks." Computer Society Press of the IEEE, 1988.

W

[Waldron, 89]
M.B. Waldron, **"Modelling of the Design Process."** "Intelligent CAD I", H. Yoshikawa, D. Gossard eds., North Holland 1989.

[Wartofsky, 68]
Marx W. Wartofsky, **"Introducción a la Filosofía de la Ciencia.."** 1968, ed. castellana Alianza Editorial,1973.

[Winograd, 75]
Terry Winograd, **"Frame Representation and the Declarative/Procedural Controversy"**, "Representation and Understanding: Studies in Cognitive Science", D.G. Bobrow, A.M. Collins eds., Academic Press, New York 1975.

[Winston, 84]
Patrick Henry Winston, **"Artificial Intelligence"**, Addison-Wesley Publishing Company Inc., 1984.

[Wirth, 80]
Nicklaus Wirth, **"Algoritmos + Estructuras de Datos = Programas"**, Ediciones del Castillo S.A., Madrid 1980.

[Woodbury & Oppenheim, 89]
R.F. Woodbury, I.J. Oppenheim, **"An Approach to Geometric Reasoning."** "Intelligent CAD I", H. Yoshikawa, D. Gossard eds., North Holland 1989.

[Wright, 51]
G.H. v. Wright, **"Deontic Logic."** Mind, vol. LX pp. 1-15, 1951.

[Wright, 63]
G.H. v. Wright, **"The Logic of Preference."** Edimburgo University Press, 1963.

X

Y

[Yuzu et al, 87]
Cai Yuzu, Liu Lujun, Wang Wei, Sun Jianwen, **"An Expert System for Automatic Allocation of 2-D Irregular Shapes."** "Expert Systems in CAD", J.S. Gero ed., North Holland 1987.

Z

[Zadeh, 65]
Lofti Zadeh, **"Fuzzy Sets."** Information and Control 8, pp. 338 y ss., 1965.

[Zhang, 89]
Zhentao Zhang, **"A Model of Conceptual Mechanical Engineering Design for Automated CAD."** Ph.D. Thesis, University of Central Florida, Orlando (Florida) 1989.

Apéndice A

Método de Representación ANA

En los apartados 2.3 y 2.4 de esta tesis se expone la necesidad de un método gráfico de representación como ayuda al desarrollo y a la especificación de proyectos de Ingeniería del Conocimiento. En este apéndice se ofrecen las bases del método ANA aquí propuesto, que es una adaptación de la metodología CORE (COntrolled Requirements Expression) descrita en [Systems Designers, 85].

A.1 - Introducción.

La Ingeniería del Conocimiento se considera aquí como la aplicación de los conocimientos de Lógica al desarrollo de sistemas automáticos de resolución de problemas. La resolución de un problema es formalmente una función entre dos espacios vectoriales, el de datos (variables de entrada) y el de soluciones (variables de salida), según muestra la figura A.1.

La resolución de un problema se realiza mediante un algoritmo, que formalmente es la ley de obtención de imágenes de la función "resolución del problema".

Un algoritmo consta de dos estructuras estrechamente relacionadas, la de variables y la de funciones. Ambas estructuras responden a la forma general de una red de árboles, es decir, un conjunto de árboles que comparten ciertas ramas.

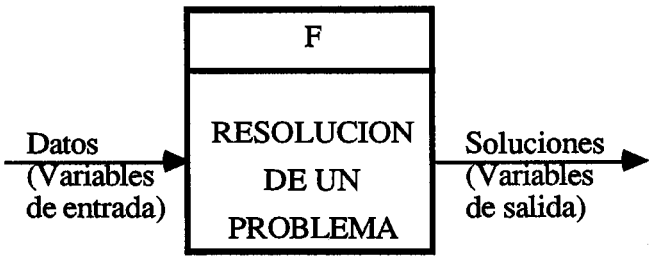


Figura A.1. La resolución de un problema como una función vectorial de variable vectorial.

El desarrollo de un algoritmo consiste en una serie de procesos sistemáticos de análisis y síntesis aplicados a la función original y a los espacios vectoriales de datos y soluciones. El objetivo es especificar por completo la red de árboles de funciones y la red de árboles de variables, así como las relaciones existentes entre ellas.

A.2 - Relaciones Estáticas: Estructura de Red de Arboles.

Las **relaciones estáticas** son las que construyen la estructura de **red de árboles**, tanto en el caso de las variables como en el de las funciones. Una estructura compleja de variables consta de estructuras de variables más elementales; dicho de otra forma, un espacio vectorial consta de subespacios vectoriales. En el caso de las funciones, una función compleja (resolución de un problema, proceso) se compone en general de funciones más elementales (subfunciones, subprocesos, resolución de subproblemas).

Un **árbol** es una estructura rígida e indivisible que puede aparecer más de una vez en la red de árboles. El **nodo raíz** representa una estructura compleja de variables (o funciones); los **nodos terminales** del árbol pueden ser variables (o funciones) elementales o bien nodos raíz de otros árboles, constituyéndose así la red de árboles.

La figura A.2 muestra la representación individual de tres árboles (A, B y C), al igual que su representación conjunta como red de árboles. Esta última no siempre será posible, pues la red de árboles puede ser demasiado compleja.

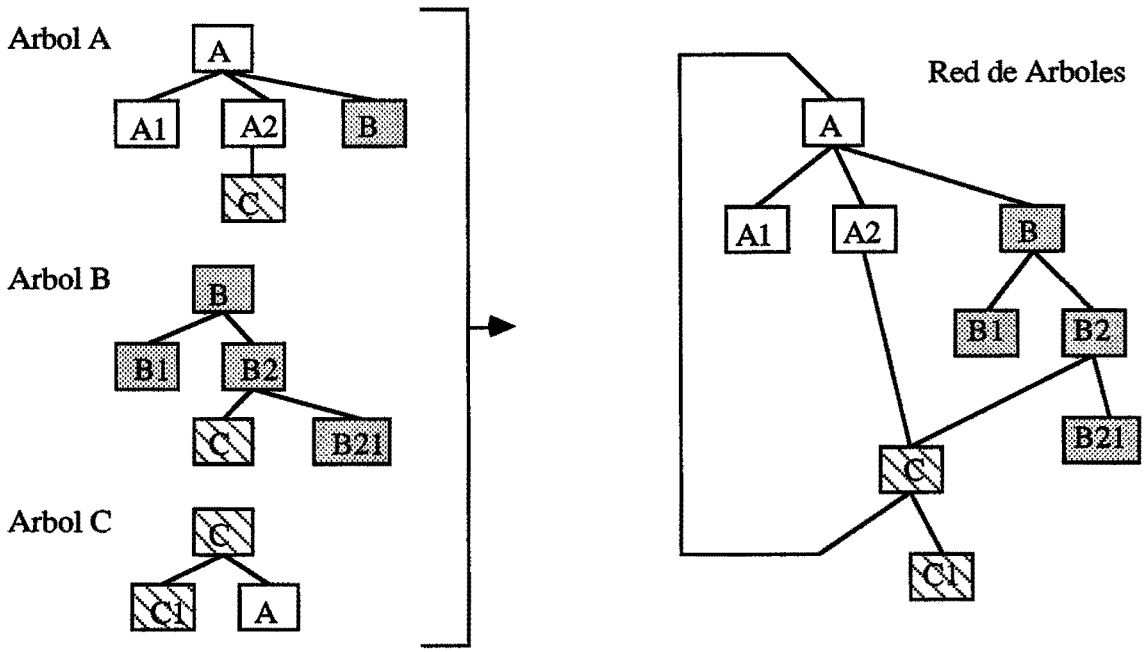


Figura A.2. Representación de relaciones estáticas: red de árboles.

Es interesante observar que la red de árboles de la figura A.2 es recursiva, pues el árbol A aparece como parte de C, y a su vez C aparece como parte de A.

A.3 - Relaciones Dinámicas: Diagramas ANA.

La representación de las relaciones estáticas es importante para organizar sistemáticamente y mantener la coherencia del conjunto de elementos de un problema. Sin embargo las **relaciones dinámicas** son las que describen el funcionamiento real del algoritmo encargado de resolver dicho problema, y por tanto su representación es indispensable para el desarrollo controlado de un proyecto de Ingeniería del Conocimiento.

Cada **diagrama ANA** muestra la estructura interna de funciones de una función compuesta y sus relaciones dinámicas, que consisten en intercambios de variables y secuencias de actuación. La figura A.3 representa la estructura interna de la función compuesta A (ver figura A.2, donde se muestran las relaciones estáticas), en la que aparecen tres subfunciones:

una función compuesta A2, una función simple A1 (ambas forman parte del árbol A) y una función compuesta B que es la raíz de otro árbol distinto.

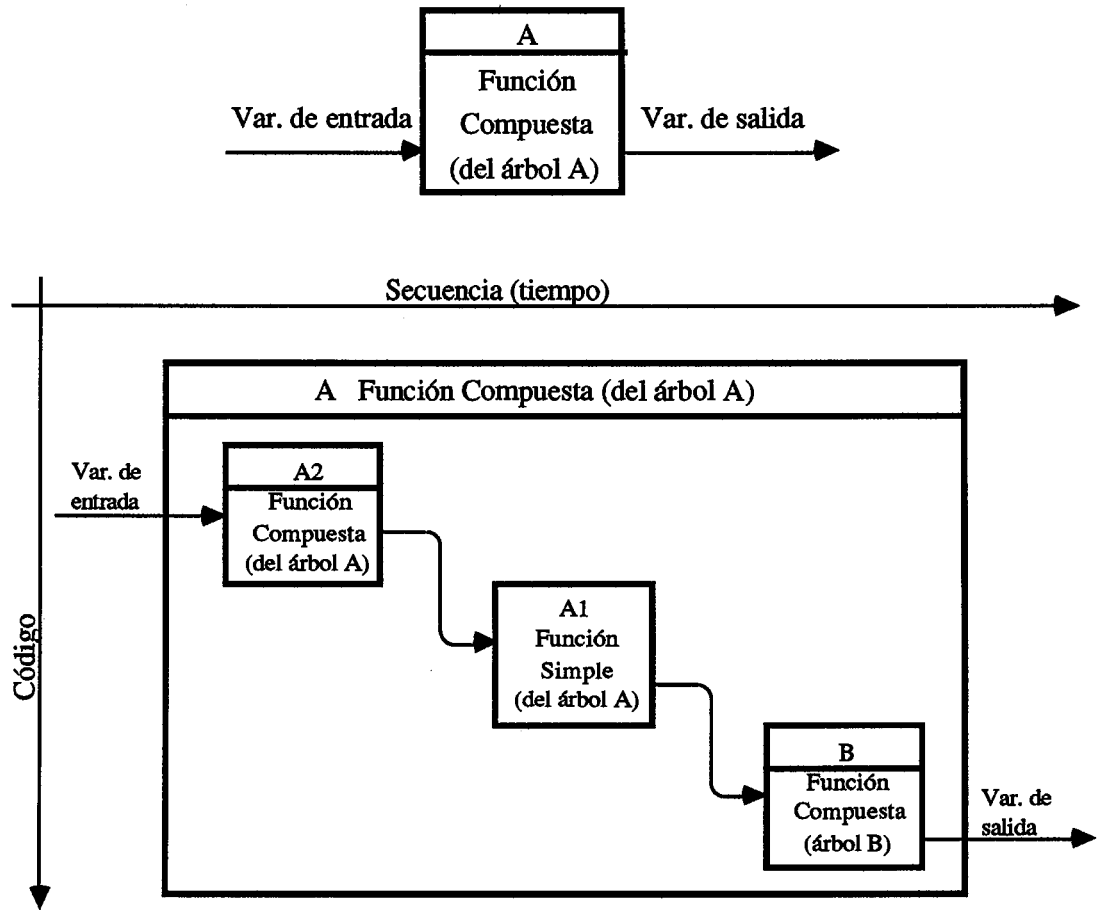


Figura A.3. Diagramas ANA.

Una función se representa por una caja con dos textos en su interior: en la parte superior aparece el **identificador** (A1, A2, etc.) y en la inferior el **nombre**, que es una frase que explica la misión de la función. En las funciones compuestas el identificador y el nombre aparecen separados por una línea que parte la caja en dos; esto indica que la estructura interna de la función se detalla en otro diagrama ANA (por ser función compuesta).

El orden horizontal de las cajas representa la **secuencia** en que actúan las funciones. El orden vertical indica la secuencia en que están escritas en el lenguaje de programación empleado. Lógicamente ambas secuencias suelen coincidir, por lo que las cajas aparecen

normalmente en diagonal; la única excepción se presenta en el caso de acciones alternativas (o bloques if) como se verá más adelante.

A.3.1 - Relaciones Dinámicas entre Variables.

Las relaciones dinámicas entre variables son tres: **ejemplificación** (o instanciación), que incluye la inicialización y la sustitución; la **composición**, y su inversa la descomposición; y por último la **dependencia** (ser función de), que es la más importante. A continuación se explican estas relaciones y se muestra cómo se representan según el método ANA:

i) **Ejemplificación**: Una variable toma un valor concreto, lo que en Lógica equivale a seleccionar un individuo concreto de entre todos los individuos que componen una clase. Casos particulares de ejemplificación son la **inicialización** (figura A.4.a) y la **sustitución** (figura A.4.b).

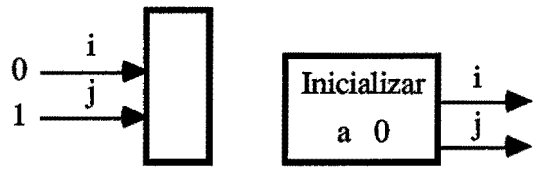


Figura A.4.a. Dos formas alternativas de representar inicializaciones.

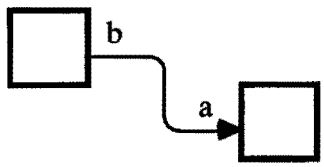


Figura A.4.b. Sustitución de un valor por otro.

ii) **Composición** (y descomposición): Una estructura de variables se separa en sus partes y viceversa (figura A.5).

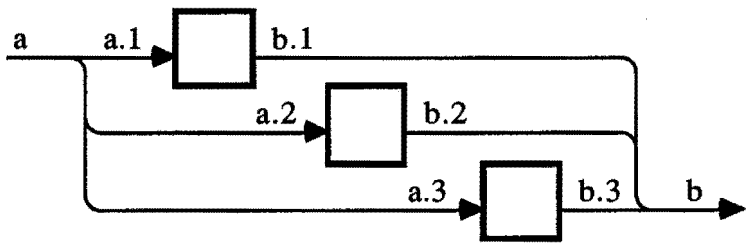


Figura A.5. Composición de la variable b y descomposición de la variable a.

iii) **Dependencia** (ser función de): Es la relación más importante y básica. Indica que el valor de un conjunto de variables puede ser hallado a partir del valor de otras por medio de una determinada función. La figura A.6 muestra cómo se representa esta relación en un caso en que dos variables (a, b) dependen de otras dos (c, d).

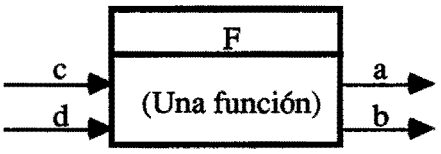


Figura A.6. Dependencia entre variables.

Un aspecto muy importante en la fase de programación es conocer el "periodo de vida" (lifetime) de una variable a la que se desea asignar memoria dinámicamente. En la figura A.7 se muestra la asignación dinámica de memoria para un vector A de n elementos, incluyendo la posterior liberación de dicha memoria.

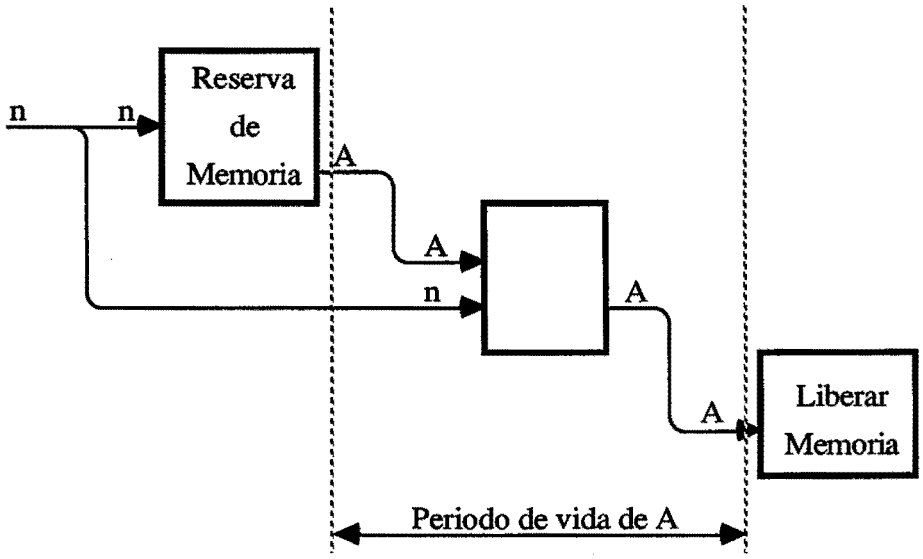


Figura A.7. Asignación dinámica de memoria y periodo de vida de una variable.

A.3.2 - Relaciones Dinámicas entre Funciones.

Las relaciones dinámicas entre funciones son cuatro: la **secuencia** de actuación; la **decisión** entre alternativas (funciones condicionadas, bloques if); la **iteración**, o bucles; y por último la **recursión**. A continuación se explican estas relaciones y su representación según el método ANA:

i) **Secuencia**: Es el orden relativo en que actúan las diferentes subfunciones que forman una función compuesta (figura A.8).

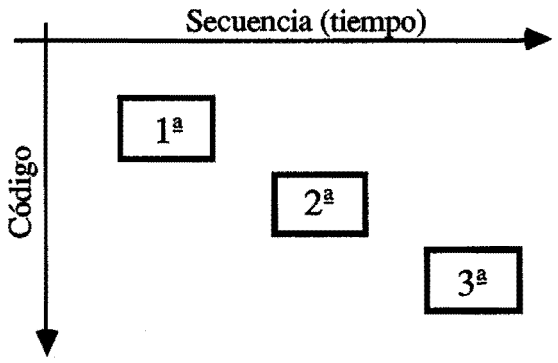


Figura A.8. Secuencia de funciones.

ii) **Decisión** (o funciones alternativas, funciones condicionadas, bloques if): Según el valor en curso de ciertas variables se aplican unas funciones u otras.

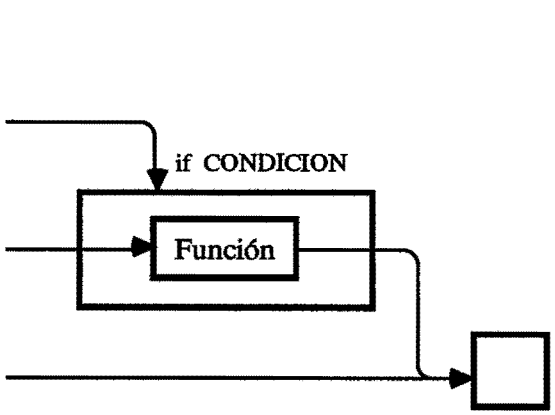


Figura A.9.a. Función condicionada.

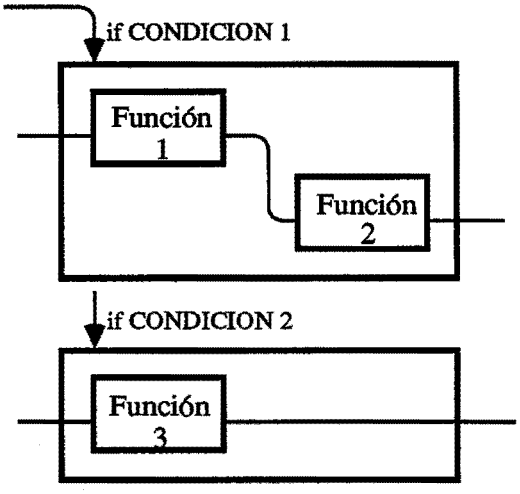


Figura A.9.b. Bloques de funciones alternativas.

La figura A.9.a muestra una función que sólo se aplica en caso de cumplirse una cierta condición, mientras que la figura A.9.b representa dos bloques de funciones alternativas; estos bloques de funciones también se llaman **macros**.

iii) **Iteraciones** (o bucles): Una iteración es una secuencia de funciones que se aplica repetidamente hasta que se cumple una cierta condición. De las variables que intervienen en el bucle algunas permanecen constantes, mientras que otras pueden modificar su valor en cada iteración. Estas últimas se llaman **orígenes** y **extremos** del bucle, y "circulan" por la **línea de retorno** del bucle. Todo bucle tiene una **condición** de fin de bucle, que puede estar situada al principio o al final de cualquier función perteneciente al mismo. Justo debajo de la condición y escrito sobre la línea de retorno hay un **texto auxiliar** que ilustra el comportamiento o la finalidad del bucle.

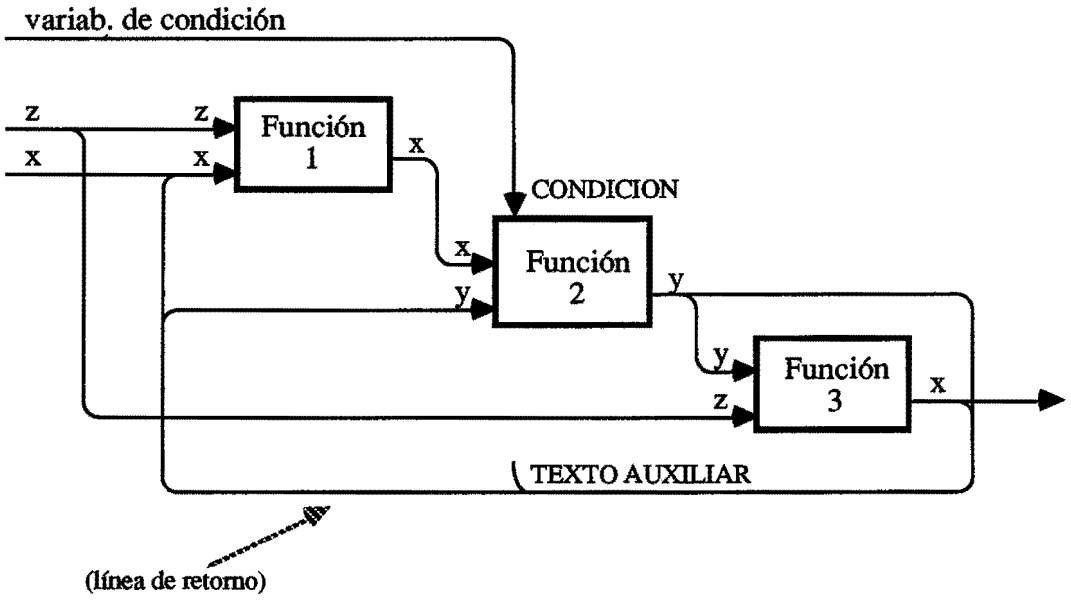


Figura A.10. Ejemplo de representación de un bucle.

La figura A.10 muestra un bucle que repite una secuencia de tres funciones, en el que la condición de fin de bucle se comprueba al principio de la segunda función. Las variables (x,y) se modifican en cada iteración, pero la variable z permanece costante durante todo el proceso.

En caso de que se desee más de una condición de fin de bucle se pueden añadir funciones condicionadas de salida de bucle, como la que aparece en la figura A.11.a. Si

ningún dato es modificado en cada iteración la línea de retorno se representa como muestra la figura A.11.b.

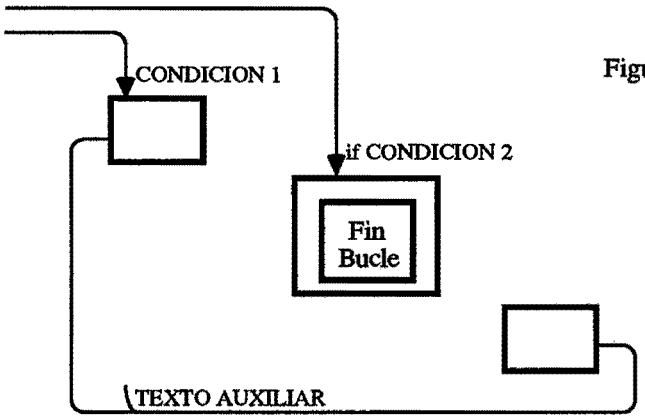


Figura A.11.a. Condición adicional de salida de bucle.

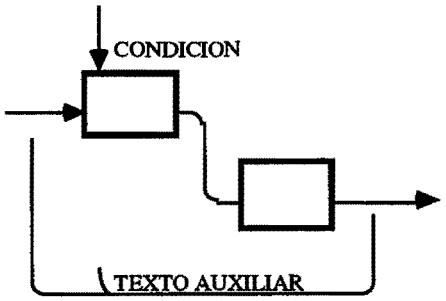


Figura A.11.b. Línea de retorno cuando ningún dato se actualiza en cada iteración.

iv) **Recursión**: La recursión se produce cuando una función aparece como parte de sí misma en algún nivel de su análisis (ver apartado A.2 Relaciones Estáticas: Red de Arboles).

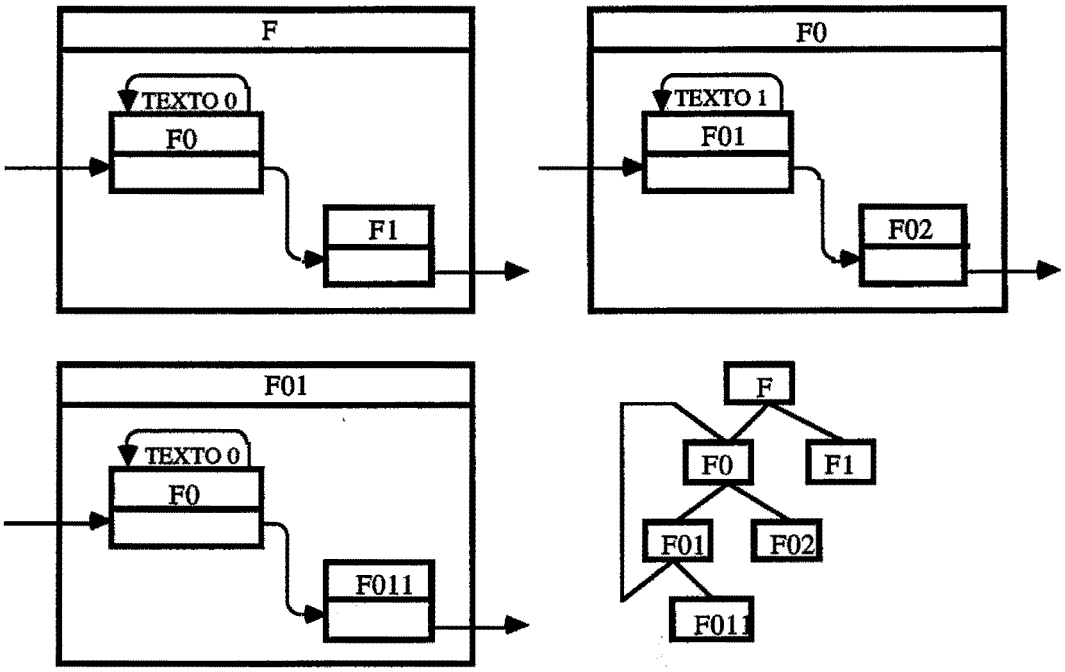


Figura A.12. Diagramas ANA y representación en árbol de un caso de recursión (F0) que aparece en el segundo nivel de anàlisis.

Una función recursiva se representa mediante una **línea de retorno** en la parte superior de la caja correspondiente a la función y un **texto auxiliar** que puede aclarar el comportamiento o la finalidad de la recursión. En la figura A.12 aparece una recursión en el segundo nivel de análisis de la función F0: obsérvese que tanto F0 como F01 son funciones recursivas.

Apéndice B

Descripción de Proyectos

En este apéndice se ofrece una descripción breve de los proyectos más importantes relacionados con la tesis. Estos proyectos son los que básicamente han dado lugar a las ideas y a la metodología expuestas en la misma, y en su mayoría corresponden a aplicaciones industriales de optimización de diseño por ordenador.

[Cuadra et al, 85]

F. Cuadra, F.J. Pérez Thoden, J.I. Pérez Arriaga, "**Optimización con Ayuda de Ordenador del Diseño de Reactancias**". ICAI, Universidad Pontificia Comillas, 1985.

[García et al, 87]

E. García, F. Cuadra, J.I. Pérez Arriaga, "**Diseño Asistido por Ordenador de Bobinas de Bloqueo**". ICAI, Universidad Pontificia Comillas, 1987.

Dentro del diseño con ayuda de ordenador existen diversas tendencias; algunas utilizan el ordenador como herramienta interactiva, llevando el control del proceso el diseñador, mientras que otras pretenden dotar de algoritmos de control y búsqueda al sistema, dejando sólo al usuario la tarea de introducir los datos. A esta última pertenece una serie de proyectos sobre diseño de diversas máquinas eléctricas (motores, transformadores y reactancias) realizados en el IIT.

Los dos proyectos fueron realizados en colaboración con Electrotecnia Arteche Hermanos S.A., dentro de un acuerdo de investigación conjunta. Vinieron motivados por la posibilidad de

un nuevo mercado (primero reactancias de limitación de cortocircuito y luego reactancias de bloqueo) y la particularidad de que estos aparatos no se producen en serie, sino de acuerdo a las necesidades del cliente. Resulta muy útil optimizar el diseño con objeto de competir con otras empresas, además de obtener presupuestos y diseños rápidamente con el menor esfuerzo posible.

El enfoque del diseño es eminentemente matemático: se representa la máquina con unos modelos que la reducen a una serie de parámetros, cuyos valores deben ser ajustados en orden a minimizar una función de mérito (coste de fabricación en este caso) y cumplir un conjunto de restricciones de diseño.

Esto se resuelve con algoritmos de optimización "ciegos" que manejan el problema en abstracto. De esta forma se puede aislar el algoritmo de optimización del tipo de elemento diseñado.

Las tareas que se realizaron fueron fundamentalmente de dos tipos:
a) Desarrollo de los modelos matemáticos que describen el comportamiento de la máquina.
b) Desarrollo de los algoritmos de optimización que son capaces de dar un diseño óptimo.

Posteriormente hubo que ajustar algoritmo y modelos para su implantación, y en este caso particular, traducir el programa de un lenguaje a otro para adaptarlo al ordenador de la empresa.

El primer proyecto (reactancias de cortocircuito) comprendió dos fases: la primera de 8 meses, que concluyó con un proyecto fin de carrera en Julio del 85, sin que los resultados fueran aprovechables directamente por la empresa (de Noviembre-84 a Junio-85); la segunda, de dos meses de duración, consistió en la adaptación de los programas al nuevo ordenador y en el refinamiento del proceso.

El segundo proyecto se realizó dos años más tarde con una duración parecida (Noviembre-86 a Junio-87 la primera fase, y hasta Septiembre-87 la segunda fase). Difiere del enfoque del primer proyecto en que, por la naturaleza de las bobinas de bloqueo, se manejan más variables discretas que continuas, por lo que los algoritmos de optimización hubieron de ser adaptados. También se puso más énfasis en el modelo térmico, en cuyo cálculo se obtuvieron resultados muy precisos.

En el segundo proyecto hubo también que emplear otro modelo electromagnético que tuviera en cuenta la influencia del grosor de los conductores, pues las bobinas de bloqueo

tienen menos arrollamientos y menos espiras, siendo apreciable el efecto del grosor de los conductores en los coeficientes de inducción.

[IIT, 87]

Instituto de Investigación Tecnológica. **"Study on Conceptual Design of Spacecrafts Using Computer-Aided Engineering Techniques"**. ESTEC contract 6886/85/NL/PP Final Report, December 1987.

Ultimamente la Agencia Espacial Europea (ESA) está realizando un importante esfuerzo para investigar las posibles aplicaciones de la Ingeniería del Conocimiento a la investigación y a la industria espacial. Esta intención se ha concretado en una serie de proyectos financiados por ESTEC (European Space Research and Technology Centre).

El Instituto de Investigación Tecnológica ha sido contratista principal en un proyecto de nueve meses de duración de título "Estudio sobre Diseño Conceptual de Satélites utilizando Técnicas de Ingeniería asistida por Ordenador". En este trabajo se realizó un estudio profundo del proceso de diseño conceptual (diseño de alto nivel o de sistemas), desde el punto de vista de la Ingeniería del Conocimiento, y se propuso un entorno informático en el que se pudiera llevar a cabo la aplicación de la Inteligencia Artificial a esta tarea.

Con estos objetivos, el proyecto se dividió en cuatro fases:

1) Modelado del proceso de diseño conceptual representando todos los flujos de información existentes.

2) Ingeniería del Conocimiento del proceso anteriormente citado.

3) Análisis de los requisitos de un entorno informático para diseño conceptual.

4) Realización de un prototipo de sistema experto de selección de órbitas.

El IIT contó con la colaboración, como subcontratistas, del Instituto de Técnica Aeroespacial (INTA) y de la Universidad Técnica de Berlín (TUB), que aportaron información sobre el proceso de diseño de sistemas espaciales. Asimismo se estuvo en contacto con el personal investigador del Flight Transportation Laboratory del MIT.

El proyecto se desarrolló en competencia con empresas europeas líderes en los sectores informático y aeroespacial, y fue sólo la primera fase de un esfuerzo más amplio, destinado a la concepción de un sistema de diseño que incorpore junto a técnicas convencionales de optimización las posibilidades de la Ingeniería del Conocimiento.

El proyecto se desarrolló entre Noviembre del 86 y Octubre del 87, con financiación de ESTEC. Posteriormente han continuado los trabajos con fondos de la CICYT durante los años 88 y 89.

[Cuadra & León, 90]

F. Cuadra, J.J. León, "**CAE Techniques: Application to Dynamic Cargo Accommodation / Location**". ESTEC Contract No. 8607/89/NL/MAC, Final Report, April 1990.

Este proyecto fue un encargo de la Agencia Espacial Europea (ESA) y se desarrolló conjuntamente entre GMV S.A. (Grupo de Mecánica de Vuelo, Sociedad Anónima) y el IIT. Surgió como una aplicación restringida de las técnicas de Ingeniería del Conocimiento a la industria espacial, y trata de desarrollar un algoritmo que optimice la distribución de carga en el transbordador espacial Hermes (a modo de caso ejemplo).

El problema planteado es muy complejo: dentro del transbordador hay varias zonas de carga, algunas presurizadas y otras no, y en cada zona hay una serie de estantes donde se alojan cajones. Los cajones pueden ser de 6 alturas estándar distintas, pero de anchura y profundidad constante, por serlo la altura y la profundidad de los estantes.

Los objetos que se deben transportar en la misión deben alojarse en los cajones teniendo en cuenta:

- Aprovechamiento máximo del espacio.
- Mínimo tiempo invertido en el proceso de carga y descarga de la nave.
- Restricciones térmicas.
- Conexiones eléctricas, de fluidos de refrigeración, etc.
- Distribución óptima de masas (posición del centro de gravedad de cada estantería).

Hay complicaciones adicionales, como por ejemplo el hecho de que parte de la carga es fija en todas las misiones y parte es propia de una misión particular; además hay carga que tiene que volver a la tierra y tiene que ser distribuida de nuevo en los cajones, presentándose restricciones en los almacenamientos intermedios.

El proyecto incluía dos tareas:

1) La especificación del concepto de una herramienta informática capaz de optimizar la distribución de las cargas y mostrar los resultados por medio de, entre otros medios, un paquete estándar de gráficos.

2) La implantación de una versión reducida de la herramienta que optimice la distribución de objetos en un sólo cajón atendiendo a varios tipos de restricciones, y muestre luego los resultados por medio de una versión avanzada de AUTOCAD.

El comienzo del proyecto tuvo lugar a finales de Noviembre de 1989, y la presentación final se celebró el 11 de Mayo de 1990. La presentación del concepto general y el funcionamiento del prototipo fueron un éxito, y actualmente el contrato se ha ampliado con una continuación de 6 meses (Septiembre de 1990 a Marzo de 1991) y con el doble de presupuesto para seguir trabajando en la misma línea.

[Cuadra et al, 90]

F. Cuadra, J.I. Pérez Arriaga, J.H. Lang, "**The Application of Knowledge Engineering to the Computer-Aided Design of Optimal Electric Drives**". International Conference on Electrical Machines, Cambridge (Massachusetts) USA, 12-15 August 1990.

Este proyecto es resultado de la colaboración entre el IIT y el MIT, y en particular con el LEES (Laboratory for Electromagnetic and Electronic Systems). Se planteó como un ejemplo de aplicación para esta tesis doctoral y como un paso previo para una aplicación informática posterior.

El problema planteado reunía todas las dificultades posibles relacionadas con variables, restricciones, atributos y módulos de cálculo. Consistía en optimizar el diseño de un motor de reluctancia variable (VRM, Variable Reluctance Motor) incluyendo el equipo electrónico (inversor) encargado de alimentarlo y el equipo encargado de su control de velocidad.

En este proyecto no se llegó a la fase de implantación informática, pero se realizó una gran labor de adquisición del conocimiento y modelado del problema, teniendo en cuenta que era un tema absolutamente desconocido. Se plantearon dos fases posibles de diseño, aparte de una fase muy interesante de análisis de consistencia y prediseño basada en modelos muy sencillos del motor.