Programación Lineal y Algoritmos Genéticos Para la Solución de un Problema de Corte

Juan David Jaramillo Jaramillo

Trabajo de grado para optar a los títulos de:
Ingeniero de Sistemas e Ingeniero Matemático

Dirigido por:
Ricardo Jaramillo Mejía
Francisco José Correa Zabala

Universidad EAFIT
Departamento de Ingeniería De Sistemas
Departamento de Ingeniería Matemática
Medellín, Colombia
Noviembre de 2008

Nota de Aceptación:

_____

_____

_____

_____

_____

Presidente del Jurado

_____

Jurado

_____

Jurado

Medellín, noviembre de 2008

“*Imagination is more important than knowledge.*”

- Albert Einstein

Agradecimientos

A mi familia, a Ani, a mis amigos...

Al finalizar este proyecto, es difícil encontrar las palabras adecuadas para expresar el sentimiento que me abarca. La experiencia de culminar satisfactoriamente dos ingenierías en simultáneo es, sin lugar a dudas, la más importante de mi vida hasta ahora, por su intensidad, por su valor, por todo lo aprendido, y por todo lo que a futuro este gran aprendizaje puede significar. A pesar de las dificultades, ahora no puedo más que expresar mi satisfacción al haber logrado el objetivo. Ahora que se acerca el final de esta importante etapa, es preciso resaltar que este largo camino no lo recorrí en solitario, son muchas las personas que participaron y estuvieron a mi lado, a quienes hoy tengo mucho que agradecer. Hoy quiero compartir con todos ellos la felicidad que me desborda.

- A *mis padres*, quienes siempre se preocuparon por darme una educación excepcional. Ellos entendieron que el mayor regalo que se les puede dar a los hijos es el conocimiento, razón por la cual nunca dudaron en estimular mi aprendizaje por medio de clases de música, arte, idiomas, etc. La elección consciente de un buen colegio y el apoyo incondicional para el estudio simultáneo de dos carreras de ingeniería en la Universidad, han sido motivación suficiente para seguir siempre adelante y terminar los estudios que hoy culmino.

- A *mis profesores*, buenos y malos, que me enseñaron mucho sobre la vida, y algunos han dejado una gran huella. De manera especial a Francisco, en quien tuve la fortuna de encontrar un buen asesor. Sus exigencias, apoyo y disponibilidad me permitieron crecer y sacar adelante este proyecto.

- A *mis hermanos y amigos*, que siempre han estado presentes, aún en la distancia. Su amistad incondicional y su compañía siempre han sido un gran apoyo.

- A *mi Ani*, una gran mujer que me ha acompañado en este proceso, quien ha soportado las dificultades y ha compartido todos mis triunfos. Durante este tiempo juntos hemos crecido mucho, y tengo la certeza de que la vida nos recompensará en grande. Te amo.

Resumen

Este proyecto de grado discute el problema de corte (roll-trim o cutting stock) en el que se busca optimizar la cantidad de material utilizada en un proceso de producción. Por la naturaleza del problema, el enfoque tradicional de la programación lineal no es muy efectivo. Una buena solución al problema debe considerar el desperdicio de material, los cambios de patrones de corte en la máquina y la cantidad de material procesado. Proponemos una solución utilizando un algoritmo genético que tiene en cuenta las consideraciones anteriores y probamos que su desempeño es superior a la solución obtenida por el enfoque como problema de programación lineal.

Palabras clave: Optimización multi-objetivo, Algoritmos genéticos, Programación lineal, Problemas de optimización, Problema de corte.

Índice General

Índice de Figuras

Índice de Tablas

Índice de algoritmos

Capítulo 1

Introducción

En los procesos industriales de producción, manejo de recursos, distribución, logística, entre otros, se busca en general maximizar las ganancias obtenidas por la empresa, mediante la optimización de dichos procesos. Por lo regular, al plantear el modelo matemático de los procesos mencionados, se llega normalmente a problemas de tipo NP-complejos. Debido a esta situación, normalmente se busca una solución lo suficientemente buena, aunque no sea la óptima global. Como ejemplo de este grupo de problemas, podemos destacar los problemas de corte y el problema de la mochila, que a su vez caracterizan una gran cantidad de problemas. De este modo, el problema de corte unidimensional tratado en este proyecto se ubica en la categoría de los problemas de corte. Los problemas de corte se han abordado por diferentes autores utilizando recocido simulado [Yen *et al.*, 2004], programación entera [Chauhan *et al.*, 2006], colonias de hormigas [Levine y Ducatelle, 2006], algoritmos genéticos [Tesharima *et al.*, 2006; Onwubolu y Mutingi, 2003; Khalifa *et al.*, 2006], entre otros. Un modelo de Optimización Matemática consiste en una función objetivo y un conjunto de restricciones en la forma de un sistema de ecuaciones o inecuaciones. Los modelos de optimización son usados en casi todas las áreas de toma de decisiones.

Aunque para la producción de cajas de cartón, por ejemplo, se realiza un corte transversal y un corte longitudinal para formar un rectángulo, el problema aquí tratado se dice unidimensional debido a que dichos cortes ocurren en instantes diferentes y son realizados por máquinas distintas. Es decir, en nuestro problema se busca una combinación de cortes lineales sobre una lámina con el fin de obtener tiras de material que minimicen el desperdicio. Este problema tiene multitud de aplicaciones dentro de distintas industrias, pues el material de la superficie sobre la que se trabaja puede ser de diversos tipos, como vidrio, metal, madera, tela, cartón, etc. Normalmente tras recibir los pedidos de los clientes y disponer del material necesario, se distribuyen los patrones de corte en el material, generalmente de forma manual. Si queremos

automatizar esta etapa del proceso de producción, el sistema a desarrollar deberá cumplir con los requisitos enunciados a continuación, para que su implementación sea realmente beneficiosa para la empresa implicada. En primer lugar, la proporción media del desperdicio de material obtenida por el sistema debe ser menor o igual que la obtenida por un humano especializado. En segundo lugar, el tiempo de ejecución para la obtención de un resultado admisible, no debe en ningún caso provocar una ralentización en el proceso global de producción, pues en tal caso, no proporcionaría ningún beneficio notable para la industria en cuestión. Por último, a la hora de resolver el problema, se deben considerar y aplicar todas las restricciones referentes al material y productos que se desea obtener. Por ejemplo, en la fabricación de las cajas de cartón la orientación del material incide en su resistencia y por tanto debe ser tenida en cuenta.

Se pretende hallar una solución óptima para el problema de corte en una dimensión (roll-trim o cutting stock) en el que se busca optimizar la cantidad de material utilizada en un proceso de producción. Este problema es un caso típico en los procesos de manufactura en el sector del cartón, metalurgia, telas, entre otros. Una buena solución al problema debe considerar el desperdicio de material, la cantidad de cambios de patrones de corte en la máquina y la cantidad de material procesado por cada patrón. Proponemos una solución utilizando algoritmos genéticos y otra solución mediante programación lineal. Posteriormente se hará una comparación de ambos métodos y se buscará un híbrido que pueda explotar las ventajas de cada uno.

El problema de corte es un problema de programación entera que se aplica en la industria para el corte de papel. El objetivo principal es recortar productos de papel de diferentes tamaños de un rollo o una lámina larga, para cumplir las órdenes o pedidos de los clientes. Un grupo de pedidos generalmente se ejecuta con desperdicio de una parte del material. El esquema óptimo de corte minimiza el desperdicio de papel a la vez que maximiza la producción. El problema involucra variables enteras en una función lineal con restricciones igualmente lineales. Dependiendo de las condiciones, el espacio de búsqueda de la solución puede ser altamente complejo.

En el caso que vamos a trabajar se cuenta con láminas de cartón de un ancho específico y una longitud que, para efectos del planteamiento teórico, se considera infinita. De estas láminas se desea recortar rectángulos para armar cajas de diferentes tamaños. Los pedidos de cajas se especifican con el ancho y largo del rectángulo, así como el número de cajas requeridas y un código de identificación de pedido. Los pedidos se introducen al sistema para generar los patrones que se utilizan para cortar el material. Con el patrón seleccionado se recorta la lámina en tiras de anchos diferentes. Sobre estas tiras se realizan posteriormente los cortes transversales de guillotina para generar los rectángulos de las cajas. La primera máquina, cuya programación se desea optimizar en nuestra investigación, cuenta con un número dado de cuchillas

que generan las tiras de cartón y eliminan los sobrantes o desperdicio. El problema consiste entonces en minimizar el desperdicio de cartón y los cambios de posición de las cuchillas, organizando de la mejor manera posible los patrones de corte que se programarán en la máquina.

La intención es obtener un modelo general y por tanto las condiciones iniciales y restricciones del problema deben ser fácilmente modificables y no generar ningún impacto negativo en el desempeño de la herramienta. Así como lo planteamos para resolver las necesidades propias de la industria de cajas de cartón, el modelo deberá ser lo suficientemente general para aplicarse en sistemas similares de otros campos o industrias.

1.1 Objetivos

1.1.1 Objetivo general

Hallar una solución óptima al problema de corte en una dimensión, con las restricciones propias del sector del cartón, pero lo suficientemente general para adaptarse a las necesidades de diferentes industrias.

1.1.2 Objetivos específicos

- Ampliar el rastreo bibliográfico de las diferentes formas como se ha enfrentado este problema, analizando sus fortalezas y debilidades bajo las condiciones particulares de nuestro proyecto.
- Mejorar el algoritmo para generar patrones de corte para la programación de una máquina de corte de cartón basado en Programación Lineal construido anteriormente.
- Mejorar el algoritmo para generar patrones de corte para la programación de una máquina de corte de cartón basado en Algoritmos Genéticos construido anteriormente.
- Realizar comparaciones entre los resultados que arrojan ambos métodos trabajados bajo diferentes entradas y situaciones.
- Utilizar nociones de manipulación de matrices dispersas y analizar el impacto que esto pueda tener en el modelo.
- Utilizar nociones de computación en paralelo y analizar los posibles beneficios que se puedan lograr en la implementación.

- Crear un algoritmo híbrido que pueda potenciar las ventajas de cada uno de los métodos implementados.
- Construir una herramienta de software que permita generar patrones de corte para la programación de una máquina de corte de cartón, de fácil manejo para un usuario final y que permita reducir los costos por desperdicio de cartón, por tiempo muerto de la máquina y por tiempos de generación de patrones de corte.
- Realizar numerosas pruebas del modelo con diferentes sets de datos, analizar y comparar los resultados para sacar conclusiones.
- Presentar diferentes alternativas de solución y posibles mejoras futuras en la solución del problema.
- Someter a evaluación para la publicación de los resultados obtenidos con la investigación en alguna revista internacional o nacional clasificada por Colciencias.

1.2 Alcance y productos

Este proyecto está dirigido al sector de industrial del país, en especial al sector de producción de cajas de cartón, donde el proyecto podría suponer grandes beneficios y un incremento en la productividad de las empresas. El proyecto está destinado a satisfacer la necesidad de tener un modelo robusto capaz de contemplar todas las posibilidades conocidas de problemas de corte aplicados en la industria local.

El proyecto, en todas sus fases y etapas, comprenderá: un análisis exhaustivo de la información básica disponible acerca del planteamiento y solución de problemas de este tipo; definición de los requisitos de los productos esperados y redefinición, si es necesario en cualquier etapa del proyecto; conocimiento de los fundamentos para el diseño de software en Matlab© y un dominio por parte del estudiante de este lenguaje; evaluación del modelo con diferentes tipos de pruebas, corriendo numerosas simulaciones y verificando que los resultados sean consistentes con la teoría y sean los esperados; y por último, el proyecto comprende la documentación de todos los procesos y fases del proyecto.

El modelo estará enmarcado bajo los lineamientos de la guía SWEBOK (Software Engineering Body of Knowledge - Cuerpo de conocimiento de la Ingeniería de Software), elaborada por la IEEE CS, que recoge los contenidos del conocimiento de ingeniería del software. La documentación se hará bajo la guía PMBOK (Project Management Body of Knowledge - Cuerpo de conocimiento de la Gerencia de Proyectos) del PMI (Project Management Institute - Instituto de la Gerencia de Proyectos) de los Estados Unidos.

Con el proyecto se pretende dar solución al problema de corte mediante un software de fácil manejo para el usuario final, adaptado a las necesidades específicas del sector del cartón. Además del software se debe entregar un manual de usuario de la herramienta y el informe de la investigación con la comparación de los métodos y el análisis de los resultados. La lista de productos esperados en el proyecto, es:

- Anteproyecto.
- Documentación recopilada de referencia.
- Diario de actividades e informes de avance.
- Trabajo escrito bajo los lineamientos de la Universidad.
- Presentación oral con material de apoyo visual.
- CD con el proyecto, la presentación y referencias bibliográficas digitales.

Las estrategias de divulgación propuestas, incluyen:

- Exposición de sustentación del trabajo de investigación ante el jurado de evaluación designado por la Universidad Eafit.
- Copia del informe final para consultas en la biblioteca de la Universidad Eafit.
- Publicación de un artículo en revistas científicas acreditadas por Colciencias.
- Exposición en foros, seminarios o congresos de uniones y sociedades relacionadas con el tema.

El problema de corte es considerado un problema de tipo NP-Difícil, que puede ser aplicado en diversos procesos industriales con diferentes condiciones y restricciones. Dentro del pensum de las carreras de Ingeniería Matemática e Ingeniería de Sistemas se encuentran temas de optimización, simulación, heurística, análisis de algoritmos, programación y muchas otras que construyen bases sólidas para afrontar este tipo de problemas.

Asimismo, el proyecto podría en un futuro beneficiar a una o varias empresas del sector manufacturero, a las que la implementación de los algoritmos aquí construidos podría significar el ahorro de grandes sumas de dinero al optimizar el proceso de corte de láminas, automatizando de la mejor manera un proceso que hasta el momento se lleva a cabo de forma manual en la mayoría de empresas locales, y eliminar al mismo tiempo la dependencia de la empresa hacia un individuo experto en la programación.

1.3 Metodología

La metodología de trabajo consiste comenzar por consultar la bibliografía disponible sobre el tema y tener un estado del arte que nos de bases sólidas para afrontar el problema. Con la información recogida, y con base en los conocimientos previos sobre procesos de optimización, programación lineal y algoritmos genéticos, se construirán los diferentes algoritmos en el lenguaje Matlab© y se harán pruebas con diferentes sets de datos. Luego se analizan los resultados y se construye sobre los algoritmos para mejorar la solución. Además se elaborará una interfaz gráfica adecuada para su fácil manejo.

Método a utilizar

El método a utilizar se basa en el ciclo PHVA presentado en la figura 1.1, el cual se fundamenta en la forma como se interrelacionan el problema, control y mejora en un proyecto para una alta satisfacción del cliente o el interesado. La etapa de *planeación* consiste principalmente en involucrar a la gente necesaria, recopilar los datos disponibles, comprender las necesidades de los clientes, estudiar exhaustivamente los procesos involucrados y desarrollar el plan de trabajo o anteproyecto. *Hacer* se refiere a establecer la mejora, verificando las causas de los problemas y recopilando los datos apropiados. En la etapa de *verificación* se analizan y se despliegan los datos; se comprueba que se hayan alcanzado los resultados deseados, se revisan los problemas y errores y se analiza qué queda aún por resolver. Por último, en la etapa de *actuación*, se documentan los anteriores procesos, se incorpora la mejora al proceso, se comunica esta mejora a los involucrados del proyecto y se identifican nuevos proyectos o problemas.

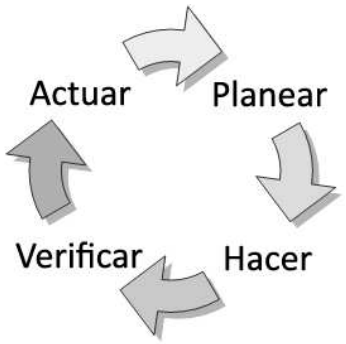


Figura 1.1: Ciclo del método PHVA

El procedimiento será realizar un estudio profundo y analítico de los conceptos y principios básicos que sobre este tema presentan los documentos bibliográficos de

referencia que se han seleccionado. Una vez se haya realizado un procesamiento de toda la información recopilada, se procede a plantear y desarrollar una propuesta de solución al problema; después se implementará la propuesta y una vez validada se repite el ciclo para encontrar posibles errores y/o mejoras. Por último se realizará una síntesis y una presentación que permita expresar, mediante gráficas y textos, las principales ideas sobre el tema de investigación.

Fase de Planificación

- Definir el tema.
- Recopilar y analizar información y bibliografía básica.
- Elaborar la propuesta del proyecto.
- Conseguir bibliografía especializada sobre el tema.
- Elaborar el primer borrador del anteproyecto de grado para revisión y comentarios del asesor y el director.
- Primera retroalimentación del asesor y director al proyecto.
- Editar la versión final del anteproyecto de grado, teniendo en cuenta la retroalimentación recibida. Entrega final del anteproyecto, para revisión y aprobación por parte de la Universidad.
- Revisar y aprobar el proyecto

Análisis y procesamiento de la información de referencia

- Complementar la información y bibliografía, si se requiere.
- Procesar y analizar con mayor profundidad la información y bibliografía recopilada.
- Analizar las experiencias adquiridas con asesores y expertos.
- Identificar las características de los proyectos que son importantes y que deben ser tenidas en cuenta para la definición y aplicación del modelo.
- Identificar los principales conceptos y principios fundamentales que deben ser reconocidos para comprender adecuadamente el modelo de Gerencia de Proyectos.

Diseño del modelo

- Definir requisitos del modelo.
- Identificar los elementos necesarios para plantear la extensión del modelo.
- Plantear la solución al problema.
- Hacer pruebas con diferentes simulaciones y sets de datos.
- Optimizar el modelo

Fase de Finalización

- Revisión y complementación del marco teórico.
- Estructuración del informe final e iniciación de su elaboración.
- Edición del informe final y entrega al jurado.
- Lectura del informe final por parte del jurado.
- Sustentación del trabajo de investigación ante el jurado.
- Publicación del trabajo de investigación.

1.4 Organización

En el capítulo 1 se hace una introducción al proyecto de grado, donde se plantean los objetivos principales y el alcance del proyecto. En el capítulo 2 se presentan los conceptos básicos más importantes para la comprensión del documento, haciendo que éste sea autocontenido. En este capítulo se explican algunas nociones básicas del álgebra lineal, de la teoría de complejidad, los problemas de optimización y los algoritmos de Programación Lineal y Algoritmos Genéticos. Además, se presenta un estudio de la computación paralela como una posibilidad de optimización de los mismos algoritmos. En el capítulo 3 se estudian los problemas de optimización, presentando algunos ejemplos de problemas clásicos y algunos de los métodos más utilizados en la solución de este tipo de problemas. En el capítulo 4 se presenta el caso de estudio y los diferentes métodos de solución planteados. En el capítulo 5 se analizan los resultados obtenidos en diferentes pruebas, y se construye un nuevo modelo a partir de estos resultados. Finalmente, en el capítulo 6 se presentan las conclusiones del proyecto y posibilidades de trabajo a futuro.

Capítulo 2

Preliminares

En este capítulo se presentan algunos elementos básicos fundamentales para lograr que el desarrollo de los temas requeridos por el proyecto sea ordenado y presentado de forma armónica y entendible, de tal forma que el documento sea autocontenido. Tras introducir una serie de nociones generales del álgebra lineal, presentamos conceptos claves en el tema de optimización en la Sección 2.2. Posteriormente se introduce en las Secciones 2.3 y 2.4 los métodos de *programación lineal* y *algoritmos genéticos* trabajados a lo largo del proyecto.

2.1 Conceptos fundamentales de álgebra lineal

Definición (Matriz). Una *matriz* es un arreglo rectangular de mn números organizados en m filas y n columnas, así:

$$\mathbf{A} = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix}$$

Ejemplo: Una matriz de 2×3.

$$\mathbf{A} = \begin{pmatrix} 5 & -2 & 0 \\ 2 & 4 & 11 \end{pmatrix}$$

Definición (Matriz cuadrada). Se dice que la matriz $\mathbf{A}$ es *cuadrada* y de orden n si $m = n$.

Ejemplo: Una matriz cuadrada de orden 2.

$$\mathbf{A} = \begin{pmatrix} 6 & 4 \\ -1 & 3 \end{pmatrix}$$

Definición (Vector). Un *vector columna* es una matriz con una sola columna y un *vector fila* es una matriz con una sola fila, es decir, una tupla ordenada de n elementos.

Ejemplo: Vector columna $\mathbf{A}$ de m elementos y un vector fila $\mathbf{B}$ de n elementos.

$$\mathbf{A} = \begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_m \end{pmatrix}, \ \mathbf{B} = \begin{pmatrix} b_1 & b_2 & \cdots & b_n \end{pmatrix}$$

Definición (Diagonal). La *diagonal* de la matriz son los elementos a_{ij} tales que $i = j$. Una matriz cuyos elementos diferentes a la diagonal son iguales a cero se llama *matriz diagonal*.

Ejemplo: Una matriz diagonal de orden 3.

$$\mathbf{A} = \begin{pmatrix} 3 & 0 & 0 \\ 0 & 4 & 0 \\ 0 & 0 & -2 \end{pmatrix}$$

Definición (Matriz identidad). La *matriz identidad* es una matriz diagonal cuyos elementos en la diagonal son todos 1 y se denota $\mathbf{I}$.

Ejemplo: Matriz identidad de orden 3.

$$\mathbf{I_3} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Definición (Transpuesta). La *transpuesta* de una matriz $\mathbf{A}$, denotada por $\mathbf{A}'$ es

$$\mathbf{A}' = \begin{pmatrix} a_{11} & a_{21} & \cdots & a_{m1} \\ a_{12} & a_{22} & \cdots & a_{m2} \\ \vdots & \vdots & \ddots & \vdots \\ a_{1n} & a_{2n} & \cdots & a_{mn} \end{pmatrix}$$

Definición (Matriz triangular). Una matriz $\mathbf{A}$ es *triangular* si $a_{ij} = 0$ para todo $i > j$ (triangular superior) o si $a_{ij} = 0$ para todo $i < j$ (triangular inferior).

Ejemplo: Matrices triangular superior $\mathbf{U}$ y triangular inferior $\mathbf{L}$ de orden 4.

$$\mathbf{U} = \begin{pmatrix} u_{11} & u_{12} & u_{13} & u_{14} \\ 0 & u_{22} & u_{23} & u_{24} \\ 0 & 0 & u_{33} & u_{34} \\ 0 & 0 & 0 & u_{44} \end{pmatrix}, \ \mathbf{L} = \begin{pmatrix} l_{11} & 0 & 0 & 0 \\ l_{21} & l_{22} & 0 & 0 \\ l_{31} & l_{32} & l_{33} & 0 \\ l_{41} & l_{42} & l_{43} & l_{44} \end{pmatrix}$$

Definición (Matriz simétrica). Una matriz $\mathbf{A}$ es *simétrica* si $\mathbf{A} = \mathbf{A}'$, es decir, $a_{ij} = a_{ji}$ para todo i,j.

Ejemplo: Matriz simétrica de orden 4.

$$\mathbf{A} = \begin{pmatrix} 3 & -2 & 0 & 4 \\ -2 & 5 & 7 & -1 \\ 0 & 7 & 3 & 9 \\ 4 & -1 & 9 & 8 \end{pmatrix}$$

Definición (Producto de matrices). Sea $\mathbf{A} = [a_{ij}]$ una matriz de $m \times n$ y sea $\mathbf{B} = [b_{ij}]$ una matriz de $n \times p$. Entonces el producto de $\mathbf{A}$ y $\mathbf{B}$ es la matriz $\mathbf{C} = [c_{ij}]$ de $m \times p$ tal que $c_{ij} = a_{i1}b_{1j} + a_{i2}b_{2j} + \cdots + a_{in}b_{nj}$. Es decir, el elemento ij-ésimo de $\mathbf{AB} = \mathbf{C}$ es igual al *producto punto* o *interno* de la i-ésima fila de $\mathbf{A}$ y la j-ésima columna de $\mathbf{B}$. El producto entre dos matrices $\mathbf{A}$ y $\mathbf{B}$ solo puede calcularse cuando el número de columnas de $\mathbf{A}$ es igual al número de filas de $\mathbf{B}$. El producto de matrices no cumple la propiedad conmutativa, es decir $\mathbf{AB} \neq \mathbf{BA}$

Ejemplo:

$$\text{sean } \mathbf{A} = \begin{pmatrix} 1 & 3 \\ -2 & 4 \end{pmatrix} \text{ y } \mathbf{B} = \begin{pmatrix} 3 & -2 \\ 5 & 6 \end{pmatrix}$$

$$\begin{aligned} \mathbf{C}_{21} &= (-2 \times 3) + (4 \times 5) \\ &= -6 + 20 \\ &= 14 \end{aligned}$$

$$\mathbf{C} = \mathbf{AB} = \begin{pmatrix} 18 & 16 \\ 14 & 28 \end{pmatrix}$$

Definición (Menor). Sean $\mathbf{A}$ una matriz de $n \times n$ y $\mathbf{M}_{ij}$ la matriz de $(n-1) \times (n-1)$ obtenida al eliminar de $\mathbf{A}$ su i-ésima fila y su j-ésima columna. A la matriz $\mathbf{M}_{ij}$ se le llama el ij-ésimo *menor* de $\mathbf{A}$.

Ejemplo: Menor de una matriz de 3×3.

$$\mathbf{A} = \begin{pmatrix} 4 & -2 & 5 \\ 1 & 4 & -1 \\ 0 & 6 & 2 \end{pmatrix}, \mathbf{M}_{32} \begin{pmatrix} 4 & 5 \\ 1 & -1 \end{pmatrix}$$

Definición (Cofactor). Sea $\mathbf{A}$ una matriz de $n \times n$. El ij-ésimo *cofactor* de $\mathbf{A}$, denotado por $\mathbf{C}_{ij}$, está dado por $\mathbf{C}_{ij} = (-1)^{i+j}|\mathbf{M}_{ij}|$, donde $|\mathbf{M}_{ij}|$ es el determinante del menor $\mathbf{M}_{ij}$.

Ejemplo: Cálculo de un cofactor de una matriz de 3×3.

$$\mathbf{A} = \begin{pmatrix} 1 & 4 & 7 \\ 3 & 0 & 5 \\ -1 & 9 & 11 \end{pmatrix}$$

$$\mathbf{C}_{23} = (-1)^{2+3}|\mathbf{M}_{23}| = -\begin{vmatrix} 1 & 4 \\ -1 & 9 \end{vmatrix} = -13$$

Definición (Determinante). Un número denominado el *determinante* de $\mathbf{A}$ es un valor que se asocia a cada matriz cuadrada y se representa como $\det(\mathbf{A})$ o $|\mathbf{A}|$. Para una matriz $\mathbf{A} = [a_{11}]$ de 1×1, $\det(\mathbf{A}) = a_{11}$. Para una matriz de 2×2, tenemos:

$$\det(\mathbf{A}) = |\mathbf{A}| = \begin{vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{vmatrix} = a_{11}a_{22} - a_{12}a_{21}$$

En general, para cualquier matriz $\mathbf{A}$ cuadrada de $n \times n$, tenemos:

$$\det(\mathbf{A}) = a_{i1}\mathbf{C}_{i1} + a_{i2}\mathbf{C}_{i2} + \cdots + a_{in}\mathbf{C}_{in}$$

para cualquier fila i. A esta fórmula se le conoce como expansión en cofactores de la fila i. Igualmente podemos expresar el determinante como una expansión en cofactores de la columna j. Para cualquier columna j, tenemos:

$$\det(\mathbf{A}) = a_{1j}\mathbf{C}_{1j} + a_{2j}\mathbf{C}_{2j} + \cdots + a_{nj}\mathbf{C}_{nj}$$

Ejemplo: Cálculo del determinante de una matriz de 3×3 por la fila 1.

$$\mathbf{A} = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$$

$$\begin{aligned} \det(\mathbf{A}) &= a_{11}(-1)^2|\mathbf{M}_{11}| + a_{12}(-1)^3|\mathbf{M}_{12}| + a_{13}(-1)^4|\mathbf{M}_{13}| \\ &= 1\begin{vmatrix} 5 & 6 \\ 8 & 9 \end{vmatrix} - 2\begin{vmatrix} 4 & 6 \\ 7 & 9 \end{vmatrix} + 3\begin{vmatrix} 4 & 5 \\ 7 & 8 \end{vmatrix} \\ &= 1(5 \times 9 - 6 \times 8) - 2(4 \times 9 - 6 \times 7) + 3(4 \times 8 - 5 \times 7) \\ &= -3 + 12 - 9 \\ &= 0 \end{aligned}$$

Definición (Adjunta). Sea $\mathbf{A}$ una matriz de $n \times n$. La *adjunta* de $\mathbf{A}$, denotada por $\text{adj}(\mathbf{A}) = \mathbf{C}^T$, es una matriz de $n \times n$ donde el ij-ésimo elemento corresponde al ji-ésimo cofactor de $\mathbf{A}$. La adjunta de $\mathbf{A}$ es una matriz de cofactores transpuesta.

Ejemplo: Adjunta de una matriz de 2×2.

$$\mathbf{A} = \begin{pmatrix} 2 & 4 \\ -1 & 9 \end{pmatrix}, \ \text{adj}(\mathbf{A}) = \begin{pmatrix} 9 & 1 \\ -4 & 2 \end{pmatrix}$$

Definición (Inversa). Sean $\mathbf{A}$ y $\mathbf{B}$ matrices de $n \times n$, tales que $\mathbf{AB} = \mathbf{BA} = \mathbf{I}$. Entonces a la matriz $\mathbf{B}$ se le llama *inversa* de $\mathbf{A}$ y se escribe $\mathbf{A}^{-1}$. Así, $\mathbf{AA}^{-1} = \mathbf{A}^{-1}\mathbf{A} = \mathbf{I}$. Si $\mathbf{A}$ tiene inversa se dice que $\mathbf{A}$ es invertible o no singular. La inversa de una matriz $\mathbf{A}$ puede calcularse como

$$\mathbf{A}^{-1} = \left(\frac{1}{\det(\mathbf{A})}\right) \text{adj}(\mathbf{A})$$

Otra forma de calcular la inversa de una matriz es mediante el uso de métodos numéricos como el método de Gauss-Jordan, resolviendo la ecuación $\mathbf{AX} = \mathbf{I}$.

Ejemplo: Inversa de una matriz de 2×2.

$$\mathbf{AA}^{-1} = \begin{pmatrix} 2 & -4 \\ 1 & 3 \end{pmatrix} \begin{pmatrix} \frac{3}{10} & \frac{4}{10} \\ -\frac{1}{10} & \frac{2}{10} \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} = \mathbf{I}_2$$

Ejemplo: Inversa de una matriz de 3×3.

$$\mathbf{AA}^{-1} = \begin{pmatrix} 2 & 0 & -1 \\ 3 & 1 & 2 \\ -1 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 1 \\ -5 & 1 & 7 \\ 1 & 0 & 2 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} = \mathbf{I}_3$$

Definición (Sistema de ecuaciones lineales). Un sistema de ecuaciones lineales donde $x_1, x_2, \ldots, x_n$ son las *variables* o incógnitas y las a_{ij} y b_j son las *constantes*, está dado por

$$\begin{array}{ccccccccc} a_{11}x_1 & + & a_{12}x_2 & + & \cdots & + & a_{1n}x_n & = & b_1 \\ a_{21}x_1 & + & a_{22}x_2 & + & \cdots & + & a_{2n}x_n & = & b_2 \\ \vdots & & \vdots & & \vdots & & \vdots & & \vdots \\ a_{m1}x_1 & + & a_{m2}x_2 & + & \cdots & + & a_{mn}x_n & = & b_m \end{array}$$

Un conjunto de ecuaciones de este tipo se denomina sistema lineal de m ecuaciones y n variables. Una *solución* para un sistema de ecuaciones lineal de m ecuaciones y n incógnitas es un conjunto de valores para las incógnitas que satisface al mismo tiempo las m ecuaciones del sistema. Un sistema de ecuaciones lineales puede ser representado en forma matricial mediante la ecuación $\mathbf{Ax} = \mathbf{b}$, dada por

$$\begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix} \times \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{pmatrix}$$

Ejemplo: Sistema lineal que se satisface con $x_1 = 1$, $x_2 = 2$.

$$\begin{array}{ccccc} x_1 & + & 2x_2 & = & 5 \\ 2x_1 & - & x_2 & = & 0 \end{array}$$

El vector $\mathbf{x} = [1\ 2]$ es una solución al sistema.

Definición (Sistema de inecuaciones lineales). Un sistema de inecuaciones lineales donde $x_1, x_2, \ldots, x_n$ son las *variables* o incógnitas y las a_{ij} y b_j son las *constantes*, está dado por

$$\begin{array}{ccccccccc} a_{11}x_1 & + & a_{12}x_2 & + & \cdots & + & a_{1n}x_n & \leq & b_1 \\ a_{21}x_1 & + & a_{22}x_2 & + & \cdots & + & a_{2n}x_n & \leq & b_2 \\ \vdots & & \vdots & & \vdots & & \vdots & & \vdots \\ a_{m1}x_1 & + & a_{m2}x_2 & + & \cdots & + & a_{mn}x_n & \leq & b_m \end{array}$$

Una *solución* para un sistema de inecuaciones lineal de m inecuaciones y n incógnitas es un conjunto de valores para las incógnitas que satisface al mismo tiempo las m inecuaciones del sistema. Un sistema de inecuaciones lineales con dos incógnitas puede ser representado en un gráfico bidimensional por medio de líneas rectas y las regiones comprendidas entre ellas, como se muestra en el ejemplo.

Ejemplo: Sistema lineal de cinco inecuaciones.

$$\begin{array}{lccccc} (a) & x_1 & & & \geq & 0 \\ (b) & & & x_2 & \geq & 0 \\ (c) & x_1 & + & x_2 & \geq & 1 \\ (d) & x_1 & - & x_2 & \leq & 1 \\ (e) & -x_1 & + & 2x_2 & \leq & 0 \end{array}$$

Este sistema lo podemos expresar en forma matricial como

$$\begin{pmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 1 \\ -1 & 1 \\ 1 & -2 \end{pmatrix} \times \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \geq \begin{pmatrix} 0 \\ 0 \\ 1 \\ -1 \\ 0 \end{pmatrix}$$

Queremos determinar el conjunto de puntos (x_1, x_2) que satisfacen las inecuaciones del sistema anterior. En primer lugar podemos observar que las dos primeras inecuaciones limitan nuestro espacio al cuadrante positivo, únicamente los puntos que contengan coordenadas no-negativas deberán ser tenidos en cuenta. Cada inecuación se satisface con los puntos ubicados en alguno de los dos espacios generados al tratar la inecuación como una ecuación. Para visualizarlo mejor, se pueden dibujar estas líneas de las igualdades con flechas que indiquen la dirección de los puntos que satisfagan la inecuación. Por ejemplo, para la inecuación *(c)* dibujamos la recta $x_1 + x_2 = 1$. Es fácil ver que el origen (0,0) no satisface la inecuación, por lo cual el espacio de puntos que satisfacen la inecuación *(c)* estará hacia arriba y hacia la derecha de la línea que hemos dibujado. En la figura 2.1 se puede observar el conjunto de inecuaciones graficadas en el plano y la región sombreada con los puntos que satisfacen el sistema de inecuaciones completo.

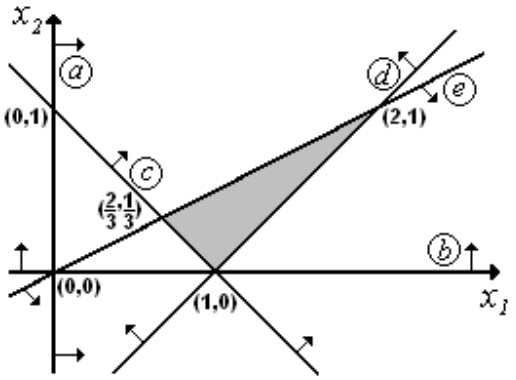


Figura 2.1: Solución a un sistema de inecuaciones lineales con dos incógnitas

Método de eliminación Gaussiana

En esta sección se trata un método eficiente para resolver un sistema de ecuaciones lineales, el método de eliminación Gaussiana o Gauss-Jordan. Por medio de este método, se muestra que cualquier sistema de ecuaciones debe cumplir uno de los siguientes casos:

1. El sistema no tiene solución.
2. El sistema tiene una sola solución.
3. El sistema tiene una cantidad infinita de soluciones.

El método de Gauss-Jordan es además importante para la definición del algoritmo Símplex, utilizado en la solución de problemas de programación lineal. Para definir el método de eliminación Gaussiana, es necesario primero definir las operaciones básicas de fila [Winston, 2005], que son la multiplicación, suma e intercambio de filas, como se explican a continuación.

Definición (Multiplicación de una fila). La multiplicación de fila se obtiene al multiplicar cualquier fila de una matriz $\mathbf{A}$ por un escalar n diferente de cero.

Ejemplo: Multiplicar por 3 la segunda fila de $\mathbf{A}$.

$$\mathbf{A} = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 1 & 3 & 5 & 6 \\ 0 & 1 & 2 & 3 \end{pmatrix}, \ \mathbf{A}' = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 3 & 9 & 15 & 18 \\ 0 & 1 & 2 & 3 \end{pmatrix}$$

Definición (Suma del múltiplo de una fila). La suma de filas se obtiene al multiplicar cualquier fila de una matriz $\mathbf{A}$ por un escalar n diferente de cero y luego

sumar el resultado con otra fila de **A**.

Ejemplo: Multiplicar por 4 la segunda fila de **A** y sumarla con la tercera fila.

$$\mathbf{A} = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 1 & 3 & 5 & 6 \\ 0 & 1 & 2 & 3 \end{pmatrix}, \ \mathbf{A'} = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 1 & 3 & 5 & 6 \\ 4 & 13 & 22 & 27 \end{pmatrix}$$

Definición (Intercambio de filas). El intercambio de filas se obtiene al intercambiar entre sí cualquier par de filas de una matriz **A**.

Ejemplo: Intercambiar la segunda fila de **A** con la tercera.

$$\mathbf{A} = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 1 & 3 & 5 & 6 \\ 0 & 1 & 2 & 3 \end{pmatrix}, \ \mathbf{A'} = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 0 & 1 & 2 & 3 \\ 1 & 3 & 5 & 6 \end{pmatrix}$$

Definición (Eliminación de Gauss-Jordan). El método de Gauss-Jordan se basa en el hecho que cualquier sucesión de operaciones básicas de fila sobre una matriz que representa un sistema de ecuaciones $\mathbf{Ax} = \mathbf{B}$, genera un sistema $\mathbf{A'x} = \mathbf{B'}$ equivalente. El método de Gauss-Jordan resuelve un sistema de ecuaciones lineales utilizando operaciones básicas de fila de modo sistemático. El método se ilustra mediante la resolución del siguiente sistema lineal:

$$\begin{array}{rcrcrcl} 2x_1 & + & 2x_2 & + & x_3 & = & 9 \\ 2x_1 & - & x_2 & + & 2x_3 & = & 6 \\ x_1 & - & x_2 & + & 2x_3 & = & 5 \end{array}$$

Representamos el sistema mediante la matriz aumentada $A|b = \mathbf{M}$, así:

$$\mathbf{A|b} = \left(\begin{array}{ccc|c} 2 & 2 & 1 & 9 \\ 2 & -1 & 2 & 6 \\ 1 & -1 & 2 & 5 \end{array}\right) = \mathbf{M}$$

En primer lugar vamos a transformar la primera columna de la matriz **M** en

$$\begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}$$

Paso 1. Multiplicar el renglón 1 de **M** por $\frac{1}{2}$. Esta operación básica nos genera

$$\mathbf{M_1} = \left(\begin{array}{ccc|c} 1 & 1 & \frac{1}{2} & \frac{9}{2} \\ 2 & -1 & 2 & 6 \\ 1 & -1 & 2 & 5 \end{array}\right)$$

Paso 2. Sumar a la fila 2 de $\mathbf{M_1}$, la fila 1 de $\mathbf{M_1}$ multiplicada por -2. El resultado de esta operación básica es

$$\mathbf{M_2} = \left(\begin{array}{ccc|c} 1 & 1 & \frac{1}{2} & \frac{9}{2} \\ 0 & -3 & 1 & -3 \\ 1 & -1 & 2 & 5 \end{array}\right)$$

Paso 3. Sumar a la fila 3 de $\mathbf{M_2}$, la fila 1 de $\mathbf{M_2}$ multiplicada por -1. El resultado de esta operación básica es

$$\mathbf{M_3} = \left(\begin{array}{ccc|c} 1 & 1 & \frac{1}{2} & \frac{9}{2} \\ 0 & -3 & 1 & -3 \\ 0 & -2 & \frac{3}{2} & \frac{1}{2} \end{array}\right)$$

La primera columna de la matriz $\mathbf{M}$ se ha transformado en

$$\left(\begin{array}{c} 1 \\ 0 \\ 0 \end{array}\right)$$

A continuación transformaremos la segunda columna de $\mathbf{M}$ en

$$\left(\begin{array}{c} 0 \\ 1 \\ 0 \end{array}\right)$$

Paso 4. Multiplicar la fila 2 de $\mathbf{M_3}$ por $-\frac{1}{3}$. El resultado de esta operación básica es

$$\mathbf{M_4} = \left(\begin{array}{ccc|c} 1 & 1 & \frac{1}{2} & \frac{9}{2} \\ 0 & 1 & -\frac{1}{3} & 1 \\ 0 & -2 & \frac{3}{2} & \frac{1}{2} \end{array}\right)$$

Paso 5. Sumar a la fila 1 de $\mathbf{M_4}$, la fila 2 de $\mathbf{M_4}$ multiplicada por -1. El resultado de esta operación básica es

$$\mathbf{M_5} = \left(\begin{array}{ccc|c} 1 & 0 & \frac{5}{6} & \frac{7}{2} \\ 0 & 1 & -\frac{1}{3} & 1 \\ 0 & -2 & \frac{3}{2} & \frac{1}{2} \end{array}\right)$$

Paso 6. Sumar a la fila 3 de $\mathbf{M_5}$, la fila 2 de $\mathbf{M_5}$ multiplicada por 2. El resultado de esta operación básica es

$$\mathbf{M_6} = \left(\begin{array}{ccc|c} 1 & 0 & \frac{5}{6} & \frac{7}{2} \\ 0 & 1 & -\frac{1}{3} & 1 \\ 0 & 0 & \frac{5}{6} & \frac{5}{2} \end{array}\right)$$

La segunda columna de la matriz $\mathbf{M}$ se ha transformado en

$$\left(\begin{array}{c} 0 \\ 1 \\ 0 \end{array}\right)$$

A continuación transformaremos la tercera columna de $\mathbf{M}$ en

$$\begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$$

Paso 7. Multiplicar la fila 3 de $\mathbf{M_6}$ por $\frac{6}{5}$. El resultado de esta operación es

$$\mathbf{M_7} = \left(\begin{array}{ccc|c} 1 & 0 & \frac{5}{6} & \frac{7}{2} \\ 0 & 1 & -\frac{1}{3} & 1 \\ 0 & 0 & 1 & 3 \end{array}\right)$$

Paso 8. Sumar a la fila 1 de $\mathbf{M_7}$, la fila 3 de $\mathbf{M_7}$ multiplicada por $-\frac{6}{5}$. El resultado de esta operación básica es

$$\mathbf{M_8} = \left(\begin{array}{ccc|c} 1 & 0 & 0 & 1 \\ 0 & 1 & -\frac{1}{3} & 1 \\ 0 & 0 & 1 & 3 \end{array}\right)$$

Paso 9. Sumar a la fila 2 de $\mathbf{M_8}$, la fila 3 de $\mathbf{M_8}$ multiplicada por $\frac{1}{3}$. El resultado de esta operación básica es

$$\mathbf{M_9} = \left(\begin{array}{ccc|c} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 2 \\ 0 & 0 & 1 & 3 \end{array}\right)$$

La matriz $\mathbf{M_9}$ representa el sistema de ecuaciones

$$\begin{aligned} x_1 \quad\quad\quad &= 1 \\ x_2 \quad\quad &= 2 \\ x_3 \quad &= 3 \end{aligned}$$

De esta forma hemos llegado finalmente a la conclusión de que el sistema $\mathbf{M_9}$ tiene solución única $x_1 = 1$, $x_2 = 2$, $x_3 = 3$. Como este sistema se obtuvo a partir de $\mathbf{M}$, entonces el sistema planteado inicialmente también tiene solución única $x_1 = 1$, $x_2 = 2$, $x_3 = 3$.

En este ejemplo básico no se ha utilizado la operación de intercambio de filas. Esta operación es de gran utilidad cuando el intercambio de dos filas de la matriz puede evitar operaciones futuras en el desarrollo del método.

2.2 Problemas de optimización

La optimización es un tema central en cualquier problema que involucre toma de decisiones, sea en ingeniería, en economía o en algún otro campo. La toma de decisiones supone escoger entre diferentes alternativas, de manera que se tome la mejor decisión. La medida de qué tan buena es cada alternativa se describe mediante una función objetivo o un índice de desempeño. La teoría y los métodos de optimización buscan seleccionar la mejor alternativa en términos de una función objetivo dada. El

área de optimización ha recibido enorme atención en los últimos años [Chong y Zak, 2001], principalmente por el rápido progreso de la tecnología en los computadores. Este campo sigue siendo muy activo en investigación, y nuevos acercamientos son propuestos constantemente. A continuación damos algunos ejemplos de problemas prácticos de optimización que se presentan en la industria.

- En una empresa de manufactura se busca producir diferentes artículos de tal manera que el costo y los tiempos de producción sean mínimos, y que el beneficio o la ganancia adquirida sea la mayor posible.
- Una empresa de transporte debe distribuir la mercancía en diferentes puntos de entrega por medio de camiones utilizando el menor tiempo posible o haciendo que los vehículos recorran la menor distancia posible.
- En una bodega o un contenedor se deben almacenar productos de diferentes formas y tamaños utilizando al máximo el espacio disponible o almacenando los productos que tengan un mayor valor.
- En un laboratorio químico se deben realizar mezclas de elementos para lograr un compuesto que cumpla con cierta condición o que genere un costo mínimo de producción.
- En una empresa manufacturera se deben recortar pequeñas piezas a partir de láminas grandes de material, de manera que en el proceso se utilice al máximo la superficie de dichas láminas.

Optimización

Un proceso de optimización consiste en encontrar la mejor opción entre un conjunto de posibilidades. En la práctica, uno puede desear el mejor o máximo (e.g., salario) o el peor o mínimo (e.g., costo). Así, la palabra óptimo puede significar máximo o mínimo según las circunstancias. Existen diferentes tratamientos para los problemas de optimización, entre los cuales se tienen [Antoniou y Lu, 2007]:

1. Métodos analíticos.
2. Métodos gráficos.
3. Métodos experimentales.
4. Métodos numéricos.

Los *métodos analíticos* se basan en las técnicas clásicas de cálculo diferencial. En estos métodos el máximo o el mínimo según un criterio de desempeño, se determinan encontrando los valores para los parámetros $x_1, x_2, \ldots, x_n$ que hacen que las derivadas

de una función $f(x_1, x_2, \ldots, x_n)$ respecto a $x_1, x_2, \ldots, x_n$ tomen valores iguales a cero. Para poder aplicar estos métodos, el problema a resolver debe ser descrito en términos matemáticos. Estos métodos pueden no necesitar el uso de un computador, pero no es posible aplicarlos siempre ni son fáciles de utilizar.

Un *método gráfico* puede ser utilizado para dibujar la función a ser optimizada si el número de variables no es mayor que dos. Si la función depende únicamente de una variable x_1, un gráfico de $f(x_1)$ versus x_1 permite observar inmediatamente el valor máximo o mínimo de la función. De manera similar, se puede construir un gráfico en tres dimensiones cuando el problema tiene dos variables. Sin embargo, el uso de los métodos gráficos es bastante limitado, puesto que la mayoría de problemas reales utilizan muchas más variables.

En ocasiones el desempeño óptimo de un sistema puede obtenerse mediante *experimentación directa*. En este método, las variables se ajustan manualmente y la función objetivo se evalúa en cada ajuste. Este método es muy inexacto y puede llevar a resultados erróneos debido al desconocimiento en la interacción de las diferentes variables.

Los métodos más importantes de optimización se basan en *métodos numéricos*, en los cuales se utilizan procedimientos numéricos iterativos para generar una serie de soluciones a partir de un estimado inicial. El proceso termina cuando un criterio de convergencia se satisface, por ejemplo cuando la variación en las variables entre una iteración y la siguiente se hacen insignificantes. Los métodos numéricos pueden usarse para resolver problemas de optimización altamente complejos, del tipo que no pueden ser resueltos analíticamente.

Definición (Optimización matemática). La optimización matemática es el estudio de los problemas donde se busca minimizar o maximizar una función real mediante la elección sistemática de valores para variables reales o enteras entre un conjunto de valores posibles. El problema puede ser planteado de la siguiente manera: dada una función $f : A \to B$ de un conjunto A a los números reales, hallar x_0 en A tal que $f(x_0) \leq f(x)$ para todo x en A (minimizar) o tal que $f(x_0) \geq f(x)$ para todo x en A (maximizar). Normalmente, A es un subconjunto del espacio Euclidiano $\Re^n$, delimitado por un conjunto de restricciones, igualdades o desigualdades que los miembros de A deben satisfacer. El dominio A de f es llamado el *espacio de búsqueda*, mientras los elementos de A son las posibles *soluciones*. La función f es la *función objetivo*, o *función de costo*. Una posible solución que minimice (o maximice) la función objetivo es llamada una *solución óptima*.

Los primeros métodos utilizados en la optimización global fueron técnicas determinísticas, basadas mayoritariamente en el principio de dividir y conquistar. Estos métodos se apoyan en la computación intensiva para explorar el espacio de soluciones

y encontrar la mejor solución posible. Así, en lugar de intentar localizar una solución óptima resolviendo un conjunto de ecuaciones por métodos algebraicos analíticos, se intentaría construir una secuencia de soluciones aproximadas que convergiera a la solución del problema al dividirlo en sub-problemas menores. Uno de los métodos más comunes que se basan en este principio es el algoritmo de Branch and Bound. Posteriormente aparecieron algoritmos estocásticos adaptativos de búsqueda aleatoria, que hacen parte de los métodos de optimización probabilísticos. En el campo de la optimización global estocástica aparecen los métodos de Recocido Simulado (Simmulated Annealing), Búsqueda Tabú (TS), Simulación de Colonias de Hormigas, entre otros. Actualmente, los algoritmos de optimización se dividen en estos dos grandes grupos: *determinísticos* y *estocásticos*.

Los métodos determinísticos [Liberti, 2008] son aquellos que pueden ser simulados por un *autómata de estado finito determinista*, es decir, una 5-tupla $(\Sigma, Q, q_0, F, \delta)$ donde Σ es el alfabeto de entrada, Q es el conjunto de estados, $q_0 \in Q$ el estado inicial, $F \subseteq Q$ el conjunto de estados finales, y $\delta : Q \times \Sigma \to Q$ la función de transición. La función de transición es tal que dado el estado actual del autómata y una palabra del alfabeto de entrada, produce el estado siguiente del autómata. El hecho de que δ sea una función bien definida nos lleva al principio de determinismo. A cualquier par (estado actual, palabra de entrada) corresponde un único estado siguiente. En otras palabras, podemos decir que un algoritmo determinístico siempre va a llevar a cabo la misma secuencia de instrucciones con los mismos valores y producir las mismas salidas ante una misma entrada.

Por otro lado, los métodos estocásticos enfrentan situaciones donde hay un elemento de aleatoriedad en la elección del siguiente paso en la computación. Las pruebas de convergencia para este tipo de algoritmos involucran teoría de la probabilidad. Debido a que los métodos estocásticos se basan en elementos de decisiones aleatorias, no se tiene la posibilidad de una garantía absoluta de encontrar la solución en un número finito de pasos. El método estocástico más simple es la búsqueda aleatoria básica, donde se generan automáticamente N soluciones a un problema dado y se selecciona la mejor de ellas. El equivalente determinístico de este método sería recorrer completamente el espacio de búsqueda para encontrar la solución óptima. Claramente esta opción no sería práctica computacionalmente, pues el tiempo para recorrer un espacio lo suficientemente complejo podría considerarse infinito, así que la opción estocástica de generar soluciones aleatoriamente y obtener una posible solución en un tiempo corto nos deja ver los beneficios que los métodos estocásticos pueden traer. Métodos como el Recocido Simulado y la Búsqueda Tabú realizan una búsqueda aleatoria dirigida y emplean diferentes técnicas para evitar caer en óptimos locales en lugar de acercarse al óptimo global.

La solución a un problema de optimización por medio de métodos determinísticos no siempre es posible ni es tarea sencilla, debido a la gran complejidad que pueden tener los espacios de búsqueda con numerosas variables y funciones no lineales. Los métodos estocásticos permiten encontrar una solución aproximada al problema en un tiempo muy inferior y que en la práctica puede ser lo suficientemente buena.

Problemas de corte

Cuando se cortan piezas pequeñas a partir de objetos grandes, surgen dos problemas: el primero es determinar las dimensiones adecuadas para los objetos grandes; y la segunda es seleccionar la forma como se van a recortar las piezas pequeñas para hacer que el desperdicio de material del objeto grande sea mínimo. Este es un proceso común en las industrias de papel, telas, madera y acero [Karelahti, 2001], donde grandes láminas del material deban recortarse en rectángulos pequeños de diferentes tamaños para satisfacer las órdenes de los clientes. El problema de minimizar el desperdicio de material se conoce como *Trim Loss Problem*, mientras que la combinación de ambos problemas se conoce como *Cutting Stock Problem*. Como el objetivo principal del corte es minimizar el desperdicio, en este documento nos referimos al problema de Trim Loss como el *Problema de Corte* en general.

El problema de corte se puede dar de diferentes formas, de acuerdo con características como la forma de los cortes, la variedad de las piezas y, la más importante, la dimensionalidad. La dimensionalidad es el número mínimo de dimensiones significativas en la determinación de una solución. En el caso más simple, solo una dimensión es relevante para la solución. Un ejemplo típico de un problema de corte unidimensional es el corte de barras de acero donde el largo de las barras grandes es fijo. Algunos autores [Karelahti, 2001] se refieren a problemas 1.5-dimensionales donde el material tiene una dimensión fija y otra variable. En los problemas bidimensionales las piezas ordenadas pueden ser ubicadas en el material en dos dimensiones libres. El caso más simple se da cuando el material y las piezas son rectangulares. En los problemas bidimensionales, el proceso de corte puede darse en diferentes pasos, donde el material es recortado en sub-láminas con ángulos rectos para luego recortar las piezas en un segundo corte. Además, cuando los cortes se extienden a través de todo el largo de la hoja, se dice que los cortes son tipo guillotina. El problema en tres dimensiones puede ser equivalente a distribuir cajas de diferentes tamaños en un contenedor, utilizando al máximo el espacio del contenedor. El problema de corte también puede darse en forma multidimensional, haciendo una abstracción del concepto básico.

La figura 2.2 ilustra el problema de corte en diferentes dimensiones. En esta y las demás figuras del documento, las áreas sombreadas en gris representan el material desperdiciado. En la figura 2.2a se muestra el proceso de corte en una dimensión,

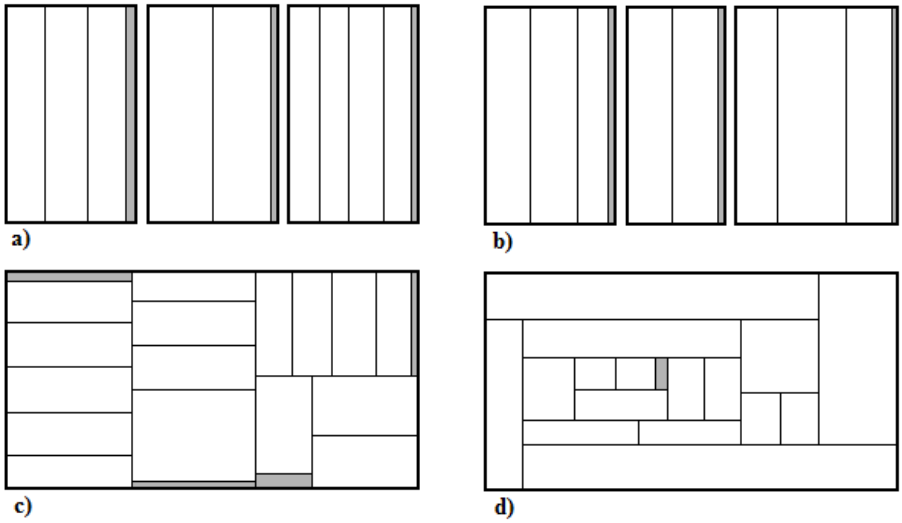


Figura 2.2: El problema de corte en diferentes dimensiones

donde se realizan únicamente cortes verticales sobre láminas de igual tamaño, mientras que el proceso de la figura 2.2b se dice 1.5-dimensional, ya que el ancho de las láminas puede variar. La figura 2.2c representa un problema de corte en dos dimensiones donde todas las piezas pueden lograrse a partir de cortes de guillotina, mientras la figura 2.2d ilustra un problema en dos dimensiones donde los cortes no son de tipo guillotina.

Debido a que incluso los problemas de corte más simples son NP-Completos, es improbable que exista un algoritmo de tiempo polinomial que lo pueda resolver. Gracias a la compleja naturaleza combinatoria del problema, puede ser considerado el problema de optimización más demandante en la industria del cartón. Diferentes métodos heurísticos han sido utilizados para obtener soluciones, pero estos métodos se utilizan en su mayoría para casos muy específicos con información particular de cada problema, por lo que es difícil utilizarlos en problemas aparentemente similares.

Aunque minimizar el desperdicio de material es el objetivo principal del problema de corte, también existen otros objetivos importantes, como los costos asociados a cambiar de un patrón de corte a otro diferente. Incluso pequeñas mejoras en el proceso de corte pueden resultar en enormes ganancias cuando la cantidad de material procesado es muy grande. El problema de corte en las empresas nacionales se resuelve de forma manual en la mayoría de los casos, generando así soluciones muy por debajo de la optimalidad.

Complejidad computacional

Dentro de la teoría de la complejidad computacional se clasifican los problemas en dos grupos: tratables e intratables. Aunque todos los problemas computables sean, en teoría, tratables, desde el punto de vista práctico las cosas son diferentes. Así, en la práctica, los problemas con límite temporal polinómico son considerados tratables y los que tienen límite temporal exponencial son considerados intratables [Gómez y Sicard, 2001]. Para analizar la complejidad de un algoritmo se utiliza la teoría de la *máquina de Turing*, un modelo computacional introducido por Alan Turing que formaliza el concepto de algoritmo. Turing construyó un modelo formal de computador, la máquina de Turing, y demostró que existían problemas que una máquina no podía resolver. La máquina de Turing consta de un cabezal lector/escritor y una cinta infinita en la que el cabezal lee el contenido, borra el contenido anterior y escribe un nuevo valor. Las operaciones que se pueden realizar en esta máquina se limitan a avanzar el cabezal lector/escritor hacia la derecha o hacia la izquierda. Para más información sobre máquinas de Turing, el lector puede remitirse a Turing [1936] o Gómez y Sicard [2001]. En el contexto de la complejidad algorítmica existe una importante clase de lenguajes que se pueden aceptar en modo determinista en tiempo polinómico, denominada la clase P.

Definición (Clase P). Un problema está en P si existe una DTM (Máquina de Turing Determinística) de complejidad polinomial que lo resuelve. La complejidad de una DTM está dada por la cantidad de movimientos de la cabeza en función del tamaño de la entrada. Por ejemplo, si determinar el camino óptimo que debe recorrer un cartero que pasa por N casas necesita menos de $50N^2 + N$ segundos, entonces el problema es resoluble en un *tiempo polinómico*. De esa manera, tiempos de $2n^2+5n$, o $4n^6+7n^4-2n^2$ son polinómicos; pero 2^n no lo es. Dentro de los tiempos polinómicos, podemos distinguir los logarítmicos ($\log(n)$), los lineales (n), los cuadráticos (n^2), cúbicos (n^3), etc.

Definición (Clase NP). En esta clase están los lenguajes que se pueden aceptar en modo no determinista en tiempo polinómico, es decir, el conjunto de problemas que pueden ser resueltos en tiempo polinómico por una máquina de Turing no determinista. La importancia de esta clase de problemas de decisión es que contiene muchos problemas de búsqueda y de optimización para los que se desea saber si existe una cierta solución o si existe una mejor solución que las conocidas. Dada su importancia, se han hecho muchos esfuerzos para encontrar algoritmos que decidan algún problema de NP en tiempo polinómico. Sin embargo, pareciera que para algunos problemas de NP (los del conjunto *NP-completo*) no es posible encontrar un algoritmo mejor que simplemente realizar una búsqueda exhaustiva. Desde el punto de vista de los problemas de decisión, es decir, aquellos problemas en los cuales se espera una respuesta *si* o *no*, la diferencia entre las clases P y NP puede ser vista como la diferencia entre

encontrar la respuesta correcta (clase P) y probar que una posible respuesta es la correcta (clase NP).

Definición (Problemas NP-Complejos). Los problemas *NP-Complejos* (o NP-Hard) comprenden la clase de problemas al menos tan difíciles como un problema NP. Un problema NP puede ser resuelto en tiempo polinómico por una máquina de Turing no determinista. Los problemas tipo NP-Duros pueden ser problemas de decisión, de búsqueda, o de optimización. Por su parte, los problemas NP-completos pueden ser descritos como los problemas en NP que tienen menos posibilidades de estar en P. Un ejemplo común de problemas NP-Complejos es el problema del agente viajero (TSP). Un ejemplo de un problema NP-Completo es el problema de la satisfactibilidad booleana.

2.3 Nociones básicas de Programación lineal

La programación lineal es un medio matemático que permite asignar una cantidad fija de recursos a la satisfacción de varias demandas, en tal forma que mientras se optimiza algún objetivo se satisfacen otras condiciones definidas [Shamblin y Stevens, 1975]. Los problemas de programación lineal comprenden la optimización de una función objetivo lineal sujeta a un conjunto de restricciones lineales de igualdad o desigualdad. En otras palabras, los problemas de programación lineal determinan la manera de alcanzar el mejor resultado (i.e. maximizar ganancia o minimizar costo) dada una lista de requerimientos representada como una ecuación lineal. Los problemas de programación lineal se pueden expresar como: *maximizar* $\mathbf{c}^T\mathbf{x}$ *sujeto a* $\mathbf{Ax}\leq\mathbf{b}$ *donde* $\mathbf{x}\geq\mathbf{0}$. El vector de *variables* es representado por $\mathbf{x}$, mientras $\mathbf{c}$ y $\mathbf{b}$ son vectores de *coeficientes* y $\mathbf{A}$ es una matriz de *coeficientes*. La expresión a maximizar o minimizar es la *función objetivo* ($\mathbf{c}^T\mathbf{x}$). Las ecuaciones $\mathbf{Ax}\leq\mathbf{b}$ son las *restricciones*. La programación lineal puede ser aplicada a diversos campos, y ha probado su utilidad en diferentes tipos de problemas de planeación, rutas, horarios, asignación, entre otros.

A manera de ejemplo, supongamos que una compañía manufacturera fabrica dos tipos de producto, A y B, y que es lo suficientemente afortunada como para vender todo lo que puede producir actualmente. Como se muestra en la tabla 2.1, cada producto requiere un tiempo de manufactura determinado en los tres departamentos. Cada departamento tiene una cantidad fija de horas-hombre disponibles por semana, como se indica en la tabla 2.3. El problema consiste en decidir qué cantidad de cada producto debe producirse con el objeto de hacer el mejor uso de los medios limitados de producción. Se supone que la ganancia por unidad de producto A es $1 y de producto B es $1,50. Se debe entonces *asignar los recursos fijos* (tiempos de manufactura por

departamento) con el propósito de *optimizar algún objetivo* (maximizar la ganancia) y *satisfacer las condiciones* definidas (no exceder las capacidades de los departamentos).

Tabla 2.1: Tiempo de producción por unidad de producto en cada departamento.

	Tiempo de producción en horas		
Producto	Depto. A	Depto. B	Depto. C
A	2	1	4
B	2	2	2

Tabla 2.2: Límites de la capacidad de producción.

Departamento	Horas-hombre por semana
A	160
B	120
C	280

Es posible expresar matemáticamente el objetivo y las restricciones. Como su nombre lo sugiere, todas las relaciones deben ser lineales. La programación lineal [Shamblin y Stevens, 1975] es una aplicación del álgebra lineal a la solución de estas ecuaciones mediante la utilización de algunas reglas especiales para asegurar que la solución satisface todas las condiciones necesarias y aun permite obtener los mejores resultados con respecto al objetivo. Lo más importante al utilizar la programación lineal es tal vez reconocer y formular el problema de manera que pueda desarrollarse y producir un objetivo para optimizar. Esto requiere imaginación y comprensión del problema y de la técnica de solución.

El primer paso en la aplicación de la programación lineal es la evaluación del objetivo. En el ejemplo se decidió el objetivo de maximizar la ganancia. Por consiguiente, es necesario evaluar el problema en términos de la ganancia. En este caso, la ganancia total es la suma de la ganancia obtenida en cada producto. Sea x_1 la variable de decisión que representa el número de unidades de producto A que se fabrican y x_2 el número de unidades del producto B. Dado que la ganancia por unidad del producto A es \$1 y por unidad del producto B es \$1,50, entonces la ecuación de la ganancia es:

$$\text{Ganancia} = C(x_1, x_2) = 1x_1 + 1,50x_2$$

Esta es la *función lineal objetivo* que debe maximizarse. Las condiciones que deben satisfacerse en este problema son aquellas que evitan exceder el tiempo disponible de producción en cada departamento. Se observa que el tiempo requerido en el departamento A depende del número de unidades que sean fabricadas de cada producto,

donde cada unidad tanto del producto A como B requiere de 2 horas, por tanto,

Tiempo de producción en A $= 2x_1 + 2x_2$

Este tiempo no debe exceder la capacidad del departamento A, donde se cuenta con 160 horas-hombre semanales. Así, la primera restricción es:

$$2x_1 + 2x_2 \leq 160$$

En un análisis similar para los departamentos B y C, tenemos que

$$\begin{aligned} x_1 + 2x_2 &\leq 120 \text{ (restricción del departamento B)} \\ 4x_1 + 2x_2 &\leq 280 \text{ (restricción del departamento C)} \end{aligned}$$

Además, puesto que no es posible fabricar un número negativo de producto A o B, estos valores deben ser mayores o iguales a cero,

$$\begin{aligned} x_1 &\geq 0 \\ x_2 &\geq 0 \end{aligned}$$

Reuniendo las ecuaciones anteriores podemos representar matemáticamente el problema de la siguiente forma:

$$\max C = x_1 + 1,5x_2$$

Sujeto a las restricciones

$$\begin{aligned} 2x_1 + 2x_2 &\leq 160 \\ x_1 + 2x_2 &\leq 120 \\ 4x_1 + 2x_2 &\leq 280 \\ x_1 &\geq 0 \\ x_2 &\geq 0 \end{aligned}$$

Las últimas dos restricciones de valores no negativos para las variables de decisión siempre están implícitas. Este problema puede expresarse en términos generales de la siguiente forma.

Sean

$$\begin{aligned} x_j &= j\text{-ésima variable de decisión} \\ c_j &= \text{coeficiente de ganancia (o costo) de la } j\text{-ésima variable} \\ z &= \text{función que debe maximizarse (o minimizarse)} \\ a_{ij} &= \text{coeficiente de la } j\text{-ésima variable en la } i\text{-ésima restricción} \\ b_i &= \text{limitación de capacidad de la } i\text{-ésima restricción} \end{aligned}$$

Por tanto, para n variables de decisión, la función objetivo se puede expresar como

$$\max z = c_1x_1 + c_2x_2 + \cdots + c_jx_j + \cdots + c_nx_n$$

Sujeto a las restricciones

$$\begin{array}{ccccccccccccc}
a_{11}x_1 & + & a_{12}x_2 & + & \cdots & + & a_{1j}x_j & + & \cdots & + & a_{1n}x_n & = & b_1 \\
a_{21}x_1 & + & a_{22}x_2 & + & \cdots & + & a_{2j}x_j & + & \cdots & + & a_{2n}x_n & = & b_2 \\
\vdots & & \vdots & & \vdots & & \vdots & & \vdots & & \vdots & & \vdots \\
a_{i1}x_1 & + & a_{i2}x_2 & + & \cdots & + & a_{ij}x_j & + & \cdots & + & a_{in}x_n & = & b_i \\
\vdots & & \vdots & & \vdots & & \vdots & & \vdots & & \vdots & & \vdots \\
a_{m1}x_1 & + & a_{m2}x_2 & + & \cdots & + & a_{mj}x_j & + & \cdots & + & a_{mn}x_n & = & b_m
\end{array}$$

Donde a_{ij}, b_i, c_j son constantes conocidas y $m < n$. En forma matricial, podemos plantear el problema de la siguiente manera: $\min \mathbf{cX}$ sujeto a $\mathbf{AX} = \mathbf{b}$ y $\mathbf{X} \geq \mathbf{0}$.

Interpretación Gráfica

Para problemas que cuenten únicamente con dos variables es posible tener una representación gráfica y obtener a partir de allí la solución. Esto proporciona una mejor comprensión del problema y facilita la interpretación de algunos pasos de la solución. Cada restricción define un área que contiene un número infinito de puntos que no exceden la desigualdad de la restricción. La figura 2.3 muestra la región factible para una respuesta, de acuerdo con las 3 restricciones planteadas.

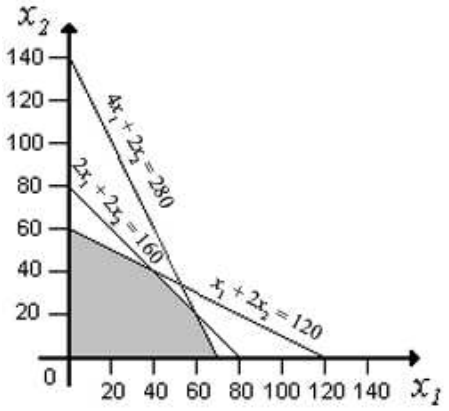


Figura 2.3: Región factible según la combinación de las restricciones.

Cualquier combinación de valores no negativos de x_1 y x_2 en dicha región es una posible solución. El problema es entonces seleccionar el punto sobre la región factible que maximice la función objetivo. Si se asigna un valor arbitrario a la función objetivo, ésta también puede graficarse, como se muestra en la figura 2.4. Como la

función objetivo es también lineal, todas las líneas tienen la misma pendiente, en diferente posición.

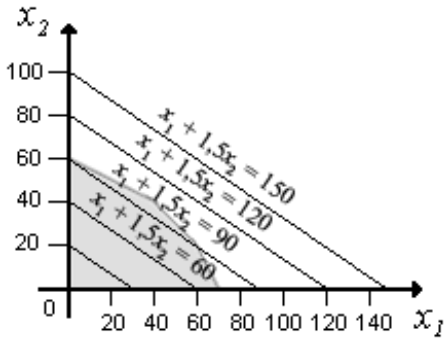


Figura 2.4: Representación gráfica de la función objetivo z.

Sobreponiendo la región factible, como se muestra en la figura 2.4, podemos determinar el máximo valor de z que se encuentra en la región factible. Según esto, podemos llegar a que la solución es:

$$\begin{aligned} x_1 &= 40 \\ x_2 &= 40 \\ z = 40 + 1,5(40) &= 100 \end{aligned}$$

La solución se encuentra en la intersección de las dos primeras restricciones, así que ambas restricciones están siendo llevadas a su límite, mientras en la tercera restricción hay una capacidad no utilizada o de holgura. Podemos verificar además que esta solución cumple todas las restricciones,

$$\begin{array}{lrcll} (1) & 2x_1 + 2x_2 & \leq & 160 & \\ & 2(40) + 2(40) & = & 160 & \text{correcto} \\ (2) & x_1 + 2x_2 & \leq & 120 & \\ & 40 + 2(40) & = & 120 & \text{correcto} \\ (3) & 4x_1 + 2x_2 & \leq & 280 & \\ & 4(40) + 2(40) & = & 240 \leq 280 & \text{correcto} \end{array}$$

Es importante comprender que si las restricciones y la función objetivo son lineales, la respuesta factible óptima debe estar en una esquina formada por la intersección de dos o más restricciones. Sólo en el caso de que la función objetivo sea paralela a una restricción es posible que el problema tenga infinitas soluciones, en cuyo caso estarán todas sobre dicha restricción, incluyendo la intersección con alguna otra restricción.

Si se utilizaran tres variables, estas podrían representarse en una gráfica tridimensional en la cual las restricciones y la función objetivo serían representadas por superficies planas. No es posible representar gráficamente el problema en más de tres

dimensiones, pero la analogía es aún válida y se dice que las restricciones y la función objetivo son hiperplanos en el espacio n-dimensional.

Solución algebraica

Como hemos visto, la solución al problema es un punto que se encuentra en la intersección de dos o más líneas, así que puede ser resuelto por medios algebraicos. Como las restricciones no son ecuaciones sino desigualdades, es necesario modificarlas para expresarlas como ecuaciones. Esto se efectúa agregando a cada restricción una variable más, denominada variable de holgura. La variable de holgura para la i-ésima restricción la vamos a representar con el término u_i. Las variables de holgura representan el valor no negativo requerido para hacer que exista igualdad en cada restricción. Si convertimos las restricciones del ejemplo anterior, tenemos

$$\begin{aligned}
&(1) \quad 2x_1 + 2x_2 + u_1 &&= 160\\
&(2) \quad x_1 + 2x_2 + u_2 &&= 120\\
&(3) \quad 4x_1 + 2x_2 + u_3 &&= 280
\end{aligned}$$

Esto constituye un sistema de ecuaciones lineales simultáneas que pueden resolverse con los medios ordinarios del álgebra. Sin embargo, en este caso hay un número infinito de soluciones, puesto que hay un total de cinco incógnitas y solo tres ecuaciones. Por lo menos dos variables deben ser iguales a cero para que exista una solución única. Este realmente era el caso de la solución gráfica presentada anteriormente. La respuesta óptima estaba en la intersección de la primera y la segunda restricción. Para estos valores $u_1 = u_2 = 0$ y $u_3 = 40$. Este valor de u_3 representa la capacidad no utilizada del departamento C. Teniendo suficiente información del problema, podríamos suponer $u_1 = u_2 = 0$ y resolver el sistema de ecuaciones

$$\begin{aligned}
2x_1 &+ 2x_2 &&= 160\\
x_1 &+ 2x_2 &&= 120\\
4x_1 &+ 2x_2 &+ u_3 &= 280
\end{aligned}$$

Haciendo una reducción del sistema por eliminación Gaussiana o algún otro método para despejar las variables, obtenemos finalmente los valores

$$\begin{aligned}
x_1 &= 40\\
x_2 &= 40\\
u_3 &= 40
\end{aligned}$$

Haciendo cero dos cualesquiera de las cinco variables en las tres ecuaciones se tendrían 10 soluciones únicas diferentes. Sin embargo, solo cinco de ellas lindan con la región factible, es decir, cumplen todas las restricciones. En este ejemplo sencillo, se podrían despejar las variables de estas cinco intersecciones y tomar la que tenga el valor máximo en la función objetivo. La tabla muestra los valores obtenidos mediante dicho procedimiento.

Tabla 2.3: Valores de las variables y la función objetivo para las posibles soluciones.

x_1	x_2	u_1	u_2	u_3	z
0	0	160	120	280	0
0	60	40	0	160	90
40	40	0	0	40	**100**
60	20	0	20	0	90
70	0	20	50	0	70

Así pues, un problema de programación lineal puede tratarse con técnicas matemáticas simples. El único inconveniente consiste en decidir cuál intersección puede ser no solamente factible, sino también dar el valor óptimo de la función objetivo. El algoritmo Símplex es una combinación del álgebra lineal y algunas reglas básicas para conducir la computación hacia una respuesta óptima que cumpla las restricciones [Shamblin y Stevens, 1975].

El método Símplex

En 1947 George Dantzig desarrolló en el Departamento de la Fuerza Aérea de los Estados Unidos el *algoritmo Símplex*, un método efectivo para resolver problemas de programación lineal [Gass, 1969]. El método Símplex de programación lineal utiliza los conceptos básicos del álgebra lineal para determinar la intersección de dos o más líneas o planos. Comienza con alguna solución factible que satisfaga todas las restricciones, y sucesivamente obtiene soluciones en las intersecciones que ofrecen mejores valores de la función objetivo. Finalmente, este método de solución proporciona un indicador que determina el punto en el cual se logra una solución óptima. El algoritmo Símplex permite encontrar una solución factible mínima en un número finito de pasos, donde cada paso o iteración consiste en encontrar una nueva solución factible cuyo valor correspondiente de la función objetivo sea mejor que el valor para la solución anterior.

Definición 2.3.1 *Una* solución factible *al problema de programación lineal es un vector* $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ *que satisface las restricciones del problema. El conjunto* Ω *de todas las soluciones factibles a un problema de programación lineal es convexo.*

Definición 2.3.2 *Una* solución factible mínima *es una solución factible que minimiza la función objetivo. El mínimo de la función objetivo se halla en un extremo del conjunto convexo* Ω *de soluciones factibles.*

Definición 2.3.3 *Un* vecindario $\aleph(S)$ *de una solución* S *es el conjunto de soluciones obtenidas al realizar un movimiento simple o una pequeña perturbación sobre la solución* S*.*

Definición 2.3.4 *Un* mínimo local *es una solución S cuyo costo es menor al de los demás elementos del vecindario $\aleph(S)$.*

Definición 2.3.5 *Un* mínimo global *es una solución S cuyo costo es menor al de todos los demás elementos del conjunto Ω de soluciones factibles.*

El siguiente procedimiento ha sido tomado de Shamblin y Stevens [1975]. En el algoritmo Símplex los datos se presentan en una matriz, donde se asigna una columna a cada variable real (las x) y una a cada variable de holgura (las u). Para cada restricción se asigna una fila. Utilizando el ejemplo anterior, obtenemos la matriz

Tabla 2.4: Matriz 1a

x_1	x_2	u_1	u_2	u_3	b
2	2	1	0	0	160
1	2	0	1	0	120
4	2	0	0	1	280

Es fácil observar que los valores de la matriz vienen de las tres restricciones. Como se explicó anteriormente, cuando se tienen n variables y m ecuaciones, con $n > m$, debemos suponer que $n - m$ variables (en este caso $5 - 3 = 2$ variables) son iguales a cero. Para un punto inicial que satisfaga las restricciones, suponemos $x_1 = x_2 = 0$. Esta es la primera solución factible, y las ecuaciones representadas en las filas de la matriz quedan reducidas a

$$\begin{aligned} u_1 &= 160 \\ u_2 &= 120 \\ u_3 &= 280 \end{aligned}$$

Las variables que están en la solución se dicen *variables básicas*, mientras que aquellas que no están en la solución, es decir las que tienen un valor igual a cero, se dicen *variables no básicas* [Shamblin y Stevens, 1975]. En el procedimiento Símplex en seguida se despeja otra intersección de las líneas de restricción que proporcione un mejor valor de la función objetivo. En términos del álgebra lineal, esto se logra suponiendo que una de las variables básicas es cero y resolviendo el nuevo sistema de ecuaciones. Para hacer esto en una forma sistemática que siempre mejore el valor de la función objetivo se requiere agregar una fila adicional a la matriz. Esta fila está formada por los coeficientes de la misma función objetivo, como se muestra en la matriz

Esta fila corresponde a la función objetivo $z = x_1 + 1.5x_2 + 0u_1 + 0u_2 + 0u_3$, que para los valores $x_1 = x_2 = 0$ toma el valor $z = 0$. En una ecuación lineal, estos

Tabla 2.5: Matriz 1b

x_1	x_2	u_1	u_2	u_3	b
2	2	1	0	0	160
1	2	0	1	0	120
4	2	0	0	1	280
1	1.5	0	0	0	0

coeficientes corresponden a las derivadas parciales de la función z.

$$\frac{\partial z}{\partial x_1} = 1,\ \frac{\partial z}{\partial x_2} = 1.5,\ \frac{\partial z}{\partial u_1} = 0,\ \frac{\partial z}{\partial u_2} = 0,\ \frac{\partial z}{\partial u_3} = 0$$

Por tanto, estos valores representan la pendiente de la función objetivo con respecto a cada una de las variables. Una pendiente positiva indica un valor creciente de la función z a medida que aumenta la variable.

En el ejemplo, la variable que presenta una mayor pendiente es x_2, por lo tanto sería más conveniente aumentar el valor de x_2, es decir, suponer $x_2 \neq 0$. La siguiente decisión consiste en determinar cuál de las variables básicas debe suponerse cero para que x_2 tenga un valor diferente de cero. Este paso requiere comprender el significado de las variables de holgura y las constantes (columna b). La columna b se obtuvo a partir de los límites de capacidad de cada departamento. Por lo tanto representa los recursos disponibles para asignar y no puede tomar valores negativos. Cuando una variable de holgura es cero, todos los recursos originalmente definidos para dicha restricción han sido asignados.

Puesto que x_2 ha sido seleccionada como la variable a aumentar, el problema consiste en cuánto podemos incrementarla. Se puede aumentar x_2 hasta alcanzar el límite máximo de la capacidad de algún departamento. Con respecto al problema original, cada unidad de x_2 requiere de 2 horas-hombre en los tres departamentos, como se puede apreciar en las restricciones del problema. Así, para el departamento 1, x_2 podría aumentarse hasta un valor máximo de $b_1/a_{12} = 160/2 = 80$ unidades. Similarmente, para las restricciones 2 y 3, el valor máximo que puede tomar x_2 es de 60 y 140 unidades respectivamente.

Después de lograr el máximo de cualquier fila, x_2 no se puede aumentar sin convertirse en no factible. Por tanto, el primer máximo que se logra define el valor limitante de x_2; en este caso la segunda restricción $x_2 = 60$ es el límite. En este punto, u_2 es ahora cero debido a que toda la holgura se ha eliminado. Finalmente debemos calcular los valores de la intersección donde $x_1 = u_2 = 0$. Una vez hecho esto, podemos actualizar la matriz formando entre x_2, u_1 y u_3 una nueva matriz identidad, es decir, están en la solución. El cómputo puede hacerse mediante operaciones de fila de la matriz para convertir x_2 en miembro de la matriz identidad.

Comenzamos por multiplicar la fila 2 por $\frac{1}{2}$ para obtener 1 en la fila clave (restricción límite) y columna clave (nueva variable básica).

Tabla 2.6: Matriz 2a

x_1	x_2	u_1	u_2	u_3	b
2	2	1	0	0	160
$\frac{1}{2}$	1	0	$\frac{1}{2}$	0	60
4	2	0	0	1	280
1	1.5	0	0	0	0

Se efectúan operaciones con las filas para obtener ceros en todos los otros espacios de la columna clave (x_2). Multiplicamos la fila 2 por -2 y la sumamos a la fila uno, luego multiplicamos la fila 2 por -2 y la sumamos a la fila 3. Finalmente multiplicamos la fila 2 por -1.5 y la sumamos a la fila z.

Tabla 2.7: Matriz 2b

x_1	x_2	u_1	u_2	u_3	b
1	0	1	-1	0	40
$\frac{1}{2}$	1	0	$\frac{1}{2}$	0	60
3	0	0	-1	1	160
$\frac{1}{4}$	0	0	$-\frac{3}{4}$	0	-90

Vemos ahora que en la fila z el valor de x_1 es $\frac{1}{4}$, es decir que por cada unidad de incremento en x_1, la función objetivo aumenta en $\frac{1}{4}$. El coeficiente de u_2 es $-\frac{3}{4}$, lo que nos indica que un aumento en las unidades de u_2 causa una disminución en el valor de la función objetivo. Esto es apenas lógico, puesto que un aumento en u_2 supone una disminución en la variable x_2, la fabricación de un producto aprovechable. Los demás coeficientes en z son cero, indicando que no puede obtenerse ganancia modificando estas variables. El -90 de la columna b representa el valor de z en la solución actual con signo cambiado. En este caso, $z = 1(0) + 1.5(60) = 90$.

Ahora nos conviene aumentar la variable x_1, puesto que es la que mayor ganancia nos genera en la función objetivo. Veamos entonces hasta dónde es posible aumentar la variable x_1. Siguiendo el mismo procedimiento anterior podemos ver que los límites actuales de x_1, son

$$(1) \quad x_1 = \frac{40}{1} = 40$$
$$(2) \quad x_1 = \frac{60}{\frac{1}{2}} = 120$$
$$(3) \quad x_1 = \frac{160}{3} = 53\frac{1}{3}$$

La fila 1 es la que tiene el menor valor permisible, por lo que podemos incrementar x_1 hasta un valor máximo de 40. Empleando el mismo procedimiento anterior,

comenzamos por dividir la fila 1 por el coeficiente de x_1. Como en este caso el valor $a_{11} = 1$, entonces no hay cambios en la matriz.

Tabla 2.8: Matriz 3a

x_1	x_2	u_1	u_2	u_3	b
1	0	1	-1	0	40
$\frac{1}{2}$	1	0	$\frac{1}{2}$	0	60
3	0	0	-1	1	160
$\frac{1}{4}$	0	0	$-\frac{3}{4}$	0	-90

A continuación multiplicamos la fila 1 por $-\frac{1}{2}$ y lo sumamos a la fila 2 para obtener un cero en la columna clave. Luego multiplicamos la fila 1 por -3 y la sumamos a la fila 3. Finalmente, multiplicamos la fila 1 por $-\frac{1}{4}$ y la sumamos a la fila z para obtener ceros en todas las posiciones de la columna clave, excepto por el 1 en la fila clave.

Tabla 2.9: Matriz 3b

x_1	x_2	u_1	u_2	u_3	b
1	0	1	-1	0	40
0	1	$-\frac{1}{2}$	1	0	40
0	0	-3	2	1	40
0	0	$-\frac{1}{4}$	$-\frac{1}{2}$	0	-100

La matriz 3b indica que se ha logrado una solución óptima debido a que la función objetivo no puede incrementarse adicionalmente mediante el incremento de cualquiera de las 5 variables. En esta solución $x_1 = 40$ y $x_2 = 40$. El valor final de la función objetivo es $z = x_1 + 1.5x_2 = 40 + 1.5(40) = 100$. Además tenemos que $u_3 = 40$, lo que indica que en el departamento C hay 40 unidades de capacidad que no se utilizan completamente.

2.4 Nociones básicas de Algoritmos genéticos

Según la teoría básica de la genética, cada individuo se compone de cromosomas y de material genético, encargados de darle las características externas e internas que lo identifican y distinguen de los demás. En el ser humano, cada célula posee 23 pares de cromosomas, donde cada cromosoma es una larga molécula de ADN. Los algoritmos genéticos modelan el fenómeno natural de herencia genética y la teoría de la evolución de Darwin, biólogo autor del libro *El Origen de las Especies.* Según Darwin, mediante la evolución biológica, todas las plantas y animales existentes descienden

de formas anteriores más primitivas. Además, la evolución se basa en la selección natural (sobreviven los más fuertes). El fenotipo es resultado de la interacción del medio ambiente en que se desarrolla un individuo y la herencia que éste recibe de sus ancestros, que va de generación en generación a través de la reproducción.

Los algoritmos genéticos (AG) son algoritmos generales de búsqueda y optimización inspirados en procesos asociados al mundo natural y la genética, que pueden aplicarse a una gran variedad de problemas. Los algoritmos genéticos fueron inventados por John Holland en los años 60 y han sido aplicados con éxito en problemas prácticos de medicina, aeronáutica, robótica, construcción, reconocimiento facial, programación, control, entre muchos otros. Un algoritmo genético [Coley, 1999] usualmente contiene:

1. Una población de posibles soluciones al problema.
2. Una forma de evaluar qué tan buena es cada posible solución.
3. Un método para combinar las soluciones y crear nuevas soluciones.
4. Un operador de mutación para evitar la pérdida de diversidad en las soluciones.

En un problema de optimización o búsqueda numérica se analiza una lista, tal vez infinita, de posibles soluciones, para encontrar la mejor solución al problema. Por ejemplo, se buscan los valores para un conjunto de variables, de tal manera que al incluirlas en un modelo matemático dado, se maximice la potencia de un automóvil. Si solamente hubieran dos parámetros ajustables, a y b, se podría generar un gran número de combinaciones y calcular para cada una la potencia p generada por dicha combinación. Si se dibuja una superficie a partir a, b y p para los ejes x, y y z, tendríamos una representación gráfica del *espacio de búsqueda* del problema. Cuando se tienen más de dos incógnitas, el espacio de búsqueda es bastante más difícil de visualizar. Sin embargo, el concepto de espacio de búsqueda sigue siendo válido siempre que se pueda definir una medida de distancia entre las soluciones y se pueda asignar un valor de *adaptación* (*fitness*) a cada solución. Las soluciones con mejor desempeño o más adaptadas ocuparán los valores más altos (picos) del espacio de búsqueda. La figura 2.5 ilustra un espacio de búsqueda que puede representar la potencia de un automóvil, de acuerdo con los parámetros a y b.

En lugar de iniciar desde un único punto del espacio de búsqueda, los *Algoritmos Genéticos* se inicializan con una *población* de posibles soluciones. Por lo general, estas soluciones se generan aleatoriamente repartidas a lo largo de todo el espacio de búsqueda. Un algoritmo típico utiliza tres operadores, *selección*, *cruce* y *mutación* (en analogía con el mundo natural) para dirigir la población (a través de una serie de pasos en el tiempo o *generaciones*) hacia la convergencia al óptimo global. Típicamente, la población se almacena como cadenas binarias de codificación de las

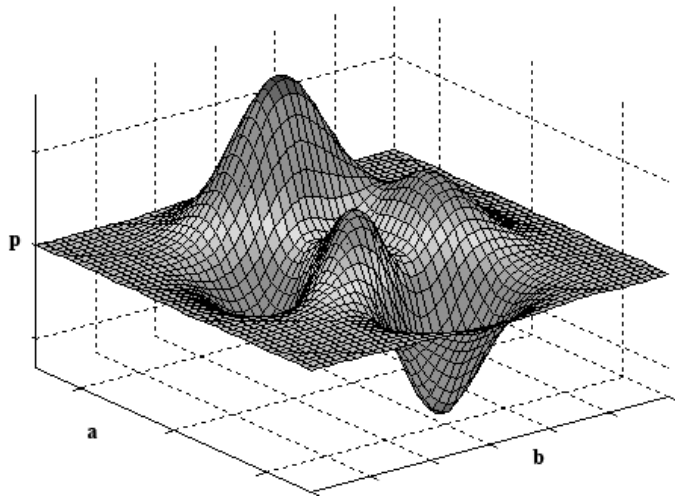


Figura 2.5: Ejemplo de espacio de búsqueda para dos variables

variables reales, aunque no necesariamente se debe utilizar dicha representación. Al igual que en el mundo natural, en los Algoritmos Genéticos los individuos nacen, mueren, se reproducen e intercambian material genético.

La *selección* busca aplicar presión sobre la población de forma similar a la selección natural en los sistemas biológicos. Los individuos con un desempeño más pobre van siendo desechados, y se conservan los individuos más *adaptados* o de mejor desempeño que tienen una probabilidad mayor de preservar su información a la siguiente generación.

El *cruce* permite a las soluciones intercambiar información de manera similar a los organismos naturales que tienen reproducción sexual. Un método común de cruce (denominado *cruce en un punto*) es escoger pares de individuos promovidos por el operador de selección, elegir de forma aleatoria un punto sobre la cadena binaria, e intercambiar la información (dígitos) a la derecha de este punto entre los dos individuos.

La *mutación* se utiliza aleatoriamente para modificar el valor de bits únicos sobre las cadenas de los individuos. La mutación se utiliza normalmente con moderación.

Después de aplicar los operadores de selección, cruce y mutación a la población inicial, una nueva población habrá sido generada y el contador generacional se incrementa en uno. Este proceso de selección, cruce y mutación se continúa por un número fijo de generaciones o hasta que se cumple algún criterio de convergencia.

Antes de aplicar algoritmos genéticos a un problema específico, es necesario tomar algunas decisiones, como el método de representación de los individuos según las variables del problema, el operador de cruce a utilizar, el tamaño de la población,

cómo aplicar el operador de mutación y el criterio de terminación.

Una forma simple del algoritmo genético básico se aprecia en el Algoritmo 1.

Algoritmo 1 Algoritmo genético básico

```
1: Generar población inicial de soluciones aleatorias
2: repetir
3:    Evaluar cada individuo según su adaptación
4:    repetir
5:       Seleccionar dos individuos con probabilidad proporcional a su adaptación
6:       Escoger un punto de cruce aleatorio y generar dos nuevos individuos
7:    hasta que Nueva población temporal generada
8:    Aplicar mutación aleatoria a la población temporal
9:    Reemplazar la población anterior por la nueva población temporal
10:   Aumentar el contador generacional
11: hasta que Número máximo de generaciones alcanzado
```

Ejemplo

A manera de ejemplo vamos a resolver el siguiente problema trivial de optimización: Hallar el valor del máximo número entero que puede ser almacenado en un espacio de memoria de 8 bits. Para este problema conocemos de antemano la solución óptima $x = 255$. En el planteamiento matemático del problema vamos a utilizar una fórmula diferente que nos lleve al mismo resultado, con el fin de mostrar que un mismo problema puede tener diferentes abstracciones y representaciones matemáticas, que nos lleven hacia el mismo objetivo. Así pues, definimos el problema matemático como encontrar el máximo valor de la función x^2 en el espacio de 0 a 255.

$$\max f(x) = x^2 \text{ ; para } x \text{ entero y } 0 \leq x \leq 255$$

En primer lugar, debemos definir la forma del algoritmo, pues como se ha dicho, los algoritmos genéticos pueden tomar diferentes formas. Para nuestro ejemplo usaremos el Algoritmo 2.

Sea la población inicial:

Algoritmo 2 Algoritmo genético para un ejemplo básico

1: Generar una población de 8 cadenas binarias aleatorias de 8 bits.
2: Decodificar cada cadena binaria a un entero x (i.e. para 00000011, $x = 3$).
3: Evaluar estos valores como soluciones al problema $f(x) = x^2$ y asignarle un valor de fitness igual a $f(x)$ a cada individuo.
4: Seleccionar la mitad con mejor fitness para avanzar a la siguiente generación.
5: Seleccionar aleatoriamente pares de estos individuos para realizar cruce en un punto. Para cada par se escoge un punto al azar y se intercambian los bits a la derecha de dicho punto entre los dos padres para crear pares de hijos.
6: Aplicar mutación aleatoriamente a los hijos intercambiando un bit al azar, con probabilidad de 1 en 10.
7: Unir los padres con los hijos para formar una nueva población de 8 individuos.
8: Volver al paso 2 y repetir hasta alcanzar 50 generaciones.

individuo	cadena	x	$f(x)$
1	00101001	41	1681
2	11000101	197	38809
3	00000101	5	25
4	10100101	165	27225
5	10011011	155	24025
6	01001000	72	5184
7	01110101	117	13689
8	10110110	182	33124

Los individuos 2, 8, 4 y 5 tienen el nivel de adaptación (fitness) más alto. Eliminando los 4 individuos restantes, obtenemos una población temporal reducida lista para el cruce.

individuo	cadena	x	$f(x)$
1	11000101	197	38809
2	10110110	182	33124
3	10100101	165	27225
4	10011011	155	24025

A continuación se escogen pares de individuos al azar. Se selecciona el 1 con el 3 y el 2 con el 4 para ser cruzados. Ahora se escoge aleatoriamente un punto sobre cada par y se intercambian los bits a la derecha para generar 4 individuos hijos. La nueva población se muestra a continuación, sin hacer uso aún del operador de mutación.

individuo	cadena	x	$f(x)$
1	11000101	197	38809
2	10110110	182	33124
3	10100101	165	27225
4	10011011	155	24025
5	10000101	133	17689
6	11100101	229	52441
7	10011110	158	24964
8	10110011	179	32041

La población inicial tiene un fitness promedio $f_{ave} = 17970$ y el mejor individuo tiene un fitness $f_{max} = 38809$. En la segunda generación el promedio ha subido a $f_{ave} = 31289$ y el mejor individuo tiene un fitness $f_{max} = 52441$. La siguiente población temporal sería:

individuo	cadena	x	$f(x)$
1	11100101	229	52441
2	11000101	197	38809
3	10110110	182	33124
4	10110011	179	32041

Esta población temporal no contiene ningún *1* como el quinto dígito de ninguno de los individuos, lo que implica que a partir de este momento ninguna cadena podrá contener dicho dígito, y así el valor máximo que puede obtenerse es 11110111 = 247. Esto nos demuestra la importancia del operador de mutación como medio para evitar la dominación por parte de cadenas sub-óptimas en la población. El operador de mutación se encarga entonces de preservar la diversidad de la población. Agregando el operador de mutación como una función para intercambiar un bit aleatoriamente en los individuos, con una probabilidad dada, se puede obtener tras varias generaciones el valor óptimo 11111111 donde $x = 255$ y $f(x) = 65025$.

Este ejemplo nos muestra la utilidad del algoritmo genético para resolver un problema sencillo como la optimización de una función con una sola variable y utilizando únicamente valores enteros. Sin embargo, la utilidad de los algoritmos genéticos va mucho más allá. Por ejemplo, mediante una transformación lineal se podría hacer un mapeo de las cadenas en el ejemplo anterior a valores reales entre -10 y 10 en intervalos de $20/255 = 0.0784$. De manera similar, para abordar problemas con más de una incógnita, se puede simplemente construir cada individuo como la concatenación de las cadenas que representan cada incógnita. Por ejemplo, si se tienen las variables x_1 y x_2, y para una solución cualquiera se tiene $x_1 = 1101$ y $x_2 = 1001$, entonces el individuo se podría representar como $a \oplus b = 11011001$. A continuación se detalla un poco más sobre los diferentes operadores del algoritmo genético.

Representación

Al trabajar con Algoritmos Genéticos lo primero que se debe definir es la forma como se representarán las posibles soluciones al problema dentro del algoritmo, es decir, los individuos. Para poder definir una representación adecuada de los individuos es importante conocer el espacio de búsqueda del problema real, y a partir de allí, construir una representación simple que permita codificar cada posible solución y hacer un mapeo del espacio de soluciones real y los individuos del algoritmo. En analogía con el mundo natural, se dice que los individuos de la población están compuestos por genes o material genético. En el caso de los algoritmos genéticos, estos genes corresponden normalmente al valor de las diferentes variables del problema para una solución dada, es decir, el individuo.

La forma de representación más utilizada en los Algoritmos Genéticos se da por medio del alfabeto binario $\langle 0, 1\rangle$, que permite pasar fácilmente del código al valor real de la solución, y aplicar diferentes operadores de manera sencilla. En el ejemplo trabajado en la sección anterior se utiliza la codificación binaria de 8 bits para representar los individuos en un espacio de búsqueda correspondiente a los enteros entre 0 y 255.

Comúnmente en un problema de Algoritmos Genéticos cada individuo es tomado como un combinación de parámetros o elementos, donde cada individuo x_i es representado como una tupla ordenada de ellos. Una ristra de genes en un conjunto de cromosomas puede ser denotada como $x_i = \{e_1, e_2, \ldots, e_l\}$, donde $e_j \in \langle 0, 1\rangle$ representa el elemento j-ésimo del individuo. El valor de l representa la cantidad de elementos o atributos. Un ejemplo de la codificación por ristra de elementos en binario se muestra a continuación.

$$\begin{array}{lccccccc} x_i = & \{ & e_1 & , & e_2 & , & \ldots & , & e_l & \} \\ & & 0101 & & 1001 & & \ldots & & 1100 & \end{array}$$

Aunque la representación en binario es la más común y la más representativa para los Algoritmos Genéticos, diferentes tipos de representación pueden ser utilizados según sea conveniente en cada problema que se intente resolver. El principal problema que surge al utilizar representaciones en base 10 o de otros tipos, es la definición de los operadores de cruce y mutación. Para seleccionar la representación hay que tener en cuenta que individuos con un grado de adaptación similar tengan una representación que también permita ver dicha similitud, pues la búsqueda de estas similitudes es la base del algoritmo. Con la codificación en binario, los individuos que representan números mayores siempre tendrán más 1s a su izquierda.

Representación Logarítmica. Para muchos problemas de ingeniería, lo único importante es la precisión relativa de las variables. En estos casos, la codificación del logaritmo de la variable puede ser más útil. Esto permite que un espacio de búsqueda muy amplio sea recorrido sin utilizar cadenas innecesariamente largas.

Codificación de Gray. Como se dijo anteriormente, los Algoritmos Genéticos pueden ser más exitosos mientras más cercana sea la codificación al espacio del problema. Por tanto, un cambio pequeño en el fenotipo (la solución) debería significar un cambio igualmente pequeño en el genotipo (la codificación). Para la representación binaria común, este no es el caso. Dado un genotipo con $l = 6$ y un fenotipo r con $0 \leq r \leq 63$, entonces $011111 = 31$ tendría que cambiar todos sus seis bits para incrementar el valor del fenotipo en uno, $100000 = 32$.

La codificación de Gray evita esto al asegurar que cada par de puntos adyacentes en el espacio de búsqueda difieran únicamente en un bit en el espacio de la representación. En la tabla 2.10 se observa esta codificación, en comparación con la codificación binaria normal para $l = 4$.

Tabla 2.10: Comparación de codificaciones binaria y de Gray.

Binaria	Gray	Binaria	Gray	Binaria	Gray	Binaria	Gray
0000	0000	0100	0110	1000	1100	1100	1010
0001	0001	0101	0111	1001	1101	1101	1011
0010	0011	0110	0101	1010	1111	1110	1001
0011	0010	0111	0100	1011	1110	1111	1000

Adaptación

Para el funcionamiento de los Algoritmos Genéticos es necesario establecer un criterio de comparación que permita decidir cuáles individuos en la población son mejores que otros. Para esto, es necesario definir, para cada individuo, una medida de desempeño o adaptación relativa a los demás individuos de la población a la que pertenece. A cada individuo de la población se le asigna una sola calificación, que se denomina función de adaptación o *fitness*, denotada por $C(x_i)$. Este valor de adaptación es necesario para tener un criterio de selección válido que permita al algoritmo converger hacia la solución óptima del problema. La función de adaptación en el Algoritmo Genético es normalmente un modelo matemático que representa la *función objetivo* del problema de optimización. En el ejemplo que vimos, la función de adaptación utilizada fue la misma función objetivo del problema de optimización, es decir $C(x) = f(x) = x^2$.

Para problemas de optimización con restricciones, donde el espacio de búsqueda deba ser reducido mediante restricciones que se aplican sobre las soluciones, se utilizan normalmente funciones de adaptación que incluyan castigos para los individuos que no cumplan las restricciones. Así, para un problema de minimización, un individuo que tenga un valor cercano al óptimo según la función objetivo, pero que no cumpla con alguna de las restricciones, será penalizado sumándole una cantidad elevada en su función de adaptación, para hacer que dicho individuo no se conserve en las siguientes

generaciones. En otros casos, estos individuos pueden ser desechados en el momento que sean generados por la naturaleza aleatoria del algoritmo.

Selección

Después de haberle dado valores de fitness a los individuos de la población, tenemos un criterio válido para comparar el desempeño de cada individuo respecto a los demás, y escoger así los mejores individuos para que avancen a la siguiente generación y puedan reproducirse y generar individuos cada vez mejor calificados. El operador de *selección* es el encargado de escoger un conjunto m de padres según su calificación, para que transmitan su información genética a la siguiente generación con el objetivo que los individuos o soluciones sean cada vez mejores. En otras palabras, el operador de selección es el encargado de hacer cumplir la ley natural de *supervivencia del más fuerte*.

El operador de selección utilizado en el primer ejemplo tomaba el mejor 50% de la población para avanzar a la siguiente generación. Este método, conocido como *selección por rango*, es práctico y simple pero no es el mejor, ya que le da la misma oportunidad a soluciones "buenas" y "muy buenas", y elimina completamente las menos buenas sin ninguna oportunidad, reduciendo la diversidad de la población. En la práctica es preferible definir un operador de selección que incluya variables estocásticas, de manera que incluso los individuos menos aptos tengan alguna posibilidad de sobrevivir, y evitar así la pérdida de diversidad en la población. Los métodos de selección más utilizados son la *selección por ruleta* y la *selección por torneo*.

Selección por ruleta. Una forma muy común del operador de selección es la selección por *ruleta* o *proporcional al fitness*. Con este método la probabilidad de selección es proporcional a la adaptación de cada individuo. La idea básica de este método es tomar la suma de las calificaciones de todos los individuos de la población como el 100% de una circunferencia, y asignar a cada miembro de la población una porción de la circunferencia, donde el tamaño de dicha porción es proporcional a su adaptación. La analogía con la ruleta viene de que se puede imaginar la circunferencia como una ruleta de casino con el agujero de cada individuo de un tamaño proporcional a su fitness, y donde a cada selección una bola se lanza para elegir un individuo según el agujero en que caiga. La probabilidad de que un individuo sea seleccionado está dada por:

$$P(i) = \frac{C(x_i)}{\sum_{j=1}^{n} C(x_j)}$$

Esta probabilidad es proporcional al nivel de adaptación de cada individuo y es no-excluyente. El proceso se repite m veces para seleccionar los m padres necesarios. Este método se ilustra en la figura 2.6 con la población inicial del primer ejemplo y el procedimiento se detalla en el algoritmo 3.

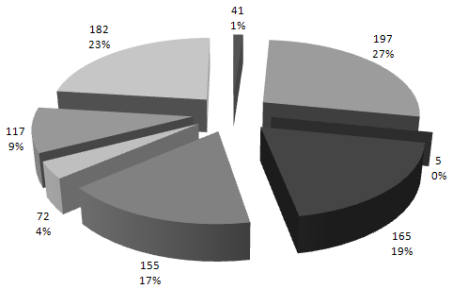


Figura 2.6: Selección por ruleta

Algoritmo 3 Operador de selección por ruleta en un AG

1: $f_{sum} = \sum f(x)$ (* la suma de los fitness de la población *)
2: **para** $j = 1$ hasta m **hacer**
3: $r = rand() \cdot f_{sum}$ (* generar un número aleatorio r en $[0, f_{sum}]$ *)
4: $f_{acum} = 0$ (* acumular fitness de la población hasta r *)
5: $i = 1$ (* contador de la población *)
6: **mientras** $f_{acum} < r$ **hacer**
7: $f_{acum} = f_{acum} + f(x_i)$ (* generar un número aleatorio r en [0,1) *)
8: $i = i + 1$
9: **fin mientras**
10: $p[j] = i - 1$ (* padre seleccionado *)
11: **fin para**

Selección por torneo. La idea del mecanismo de selección por torneo es tomar dos individuos de forma aleatoria, donde cada individuo tiene la misma probabilidad de ser elegido. Posteriormente se realiza una competencia o torneo entre ambos, y el individuo con mejor fitness es el vencedor y se selecciona para continuar como padre de la siguiente generación. Este método es bastante sencillo y permite a todos los individuos de la población tener una cierta probabilidad de ser seleccionados. El algoritmo 4 presenta una forma de selección por torneo.

Algoritmo 4 Operador de selección por torneo en un AG

1: **para** $j = 1$ hasta m **hacer**
2: $a = rand(1, n)$ (* generar un número aleatorio a en $[0, n]$ *)
3: $b = rand(1, n)$ (* generar un número aleatorio b en $[0, n]$ *)
4: **si** $C(x_a) > C(x_b)$ **entonces**
5: $p[j] = a$ (* padre a seleccionado *)
6: **si no**
7: $p[j] = b$ (* padre b seleccionado *)
8: **fin si**
9: **fin para**

Elitismo. La selección por ruleta o por torneo no garantizan la selección de ningún individuo en particular, ni siquiera del mejor. En ocasiones este mejor individuo puede no ser elegido y perderse para la siguiente población, arrojando a la basura la mejor solución al problema que se tenía. Para evitar perder la mejor solución o incluso un grupo de las mejores soluciones se puede implementar un operador de elitismo. El *elitismo* se puede implementar haciendo una copia exacta del mejor individuo hacia la siguiente generación sin aplicarle siquiera la mutación. En ocasiones el elitismo puede utilizarse como un porcentaje que asegure el paso de los mejores ϵ individuos intactos o pasando por la mutación.

Cruce

Uno de los factores claves de la evolución de las especies está en la posibilidad de transmitir las propiedades genéticas de los individuos a sus descendientes, compartiendo la información genética de los padres en la creación de un nuevo individuo. En los Algoritmos Genéticos, el operador de cruce se encarga de combinar soluciones para obtener nuevos individuos que compartan información de dos individuos seleccionados de la generación anterior. A los individuos generados se les llama comúnmente *hijos* y los individuos a partir de los cuales fueron generados se les llama *padres*. El cruce puede implementarse de diferentes formas, de las cuales se explican algunas a continuación.

Cruce en un punto. La forma más simple del operador de cruce es el cruce en un solo punto, donde se elige un bit al azar y se intercambian los bits a la derecha del mismo entre los dos padres para generar dos nuevos individuos hijos. Por ejemplo, si el punto de corte es el bit 4, tenemos:

padres	hijos
11100101	1110|1111
11001111	1100|0101

Cruce en dos puntos. Otra forma de definir el operador de cruce es el cruce en dos puntos, en el que se eligen dos bits al azar y se intercambian los bits que hay entre ellos en los dos padres para generar dos nuevos individuos hijos. Por ejemplo, si los puntos de corte son el bit 3 y el 7, tenemos:

padres	hijos
11100101	111|011|01
11001111	110|001|11

Cruce uniforme. El cruce uniforme hace que cada bit tenga una probabilidad p de ser tomado de un padre o del otro. Para cada bit se escoge aleatoriamente el padre del cual será tomado. Por consiguiente, el otro hijo tomará los bits de sus padres en forma contraria. Así, suponiendo que se tomen los bits con una máscara 11010110, donde un 1 significa que se toma el bit del primer padre y un 0 significa que se toma del segundo, tenemos:

padres	hijos
11100101	11001101
11001111	11100111

En la figura 2.7 se aprecia una representación gráfica de los diferentes tipos de operadores de cruce que hemos explicado,(a) en un punto, (b) en dos puntos y (c) uniforme.

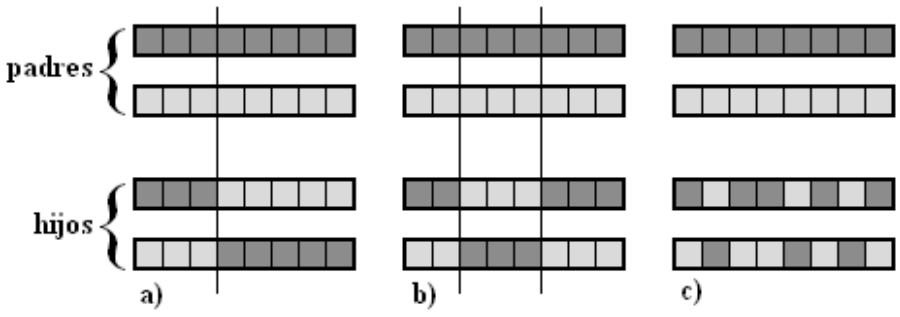


Figura 2.7: Tipos de operadores de cruce por (a) un punto, (b) dos puntos y (c) uniforme.

Mutación

El operador de mutación le da a los algoritmos genéticos la posibilidad de que en ocasiones no comunes y con cierta probabilidad P_m se produzcan cambios o alteraciones en los individuos de la población, logrando recorrer por completo el espacio de búsqueda y haciendo que el algoritmo no se centre en un individuo controlador que supondrá un mínimo local y por el contrario, mantenga exploración del dominio del problema, lo que le permite ir mejorando y variando al individuo controlador hasta llegar a la solución ideal o cercana.

Para los algoritmos genéticos con representación binaria este operador es bastante simple de implementar. En cada generación, la población completa se repasa intercambiando ocasionalmente un bit, es decir cambiar un 1 por un 0 o viceversa. La probabilidad de mutación P_m es normalmente del orden de 0.001, es decir, se cambia uno de cada mil bits, aunque al igual que todos los operadores del algoritmo genético, la correcta definición del operador depende del problema.

				Mutación en el bit 4				
individuo x_i	1	1	0	1	0	0	1	0
				↓				
mutación de x_i	1	1	0	0	0	0	1	0

La mutación es una herramienta muy poderosa pero se debe tener cuidado de no hacer un uso exagerado de ella, ya que esto puede hacer que la población tenga mucha variabilidad y el algoritmo se quede saltando de un lugar a otro sin realmente establecer mejoras y sin alcanzar la convergencia, transformando el algoritmo genético en un método de fuerza bruta. El algoritmo 5 muestra un método sencillo para aplicar este operador en un algoritmo genético con una población de N individuos, cada uno con L genes o bits.

Algoritmo 5 Operador de mutación en un algoritmo genético

```
1: para i = 1 hasta N hacer
2:   para j = 1 hasta L hacer
3:     r = rand()      (* generar un número aleatorio r en [0,1) *)
4:     si r ≤ P_m entonces
5:       x_i(j) = 1 − x_i(j)      (* intercambiar gen j del individuo i *)
6:     fin si
7:   fin para
8: fin para
```

Al igual que los demás operadores de los Algoritmos Genéticos, la mutación puede efectuarse de diferentes maneras. En la forma más simple, que ya hemos explicado, se intercambia el valor de algunos bits seleccionados al azar. Cuando se utiliza representación binaria, una mutación en el primer bit es mucho más significativa

que una mutación en el último bit. Por ejemplo, para $l = 8$, si se aplica la mutación sobre el individuo $x = 00010111 = 23$ en el primer bit, el resultado sería $x = 10010111 = 151$, mientras que si la mutación ocurre en el último bit, el resultado sería $x = 00010110 = 22$. Este resultado nos indica que sobre el final de la ejecución del algoritmo, cuando la población está cerca de alcanzar los valores óptimos, sería conveniente restringir la mutación a los bits menos significativos. De igual forma, podría convenir aplicar la mutación sobre los bits más significativos durante las primeras generaciones, con el fin de asegurar una exploración amplia del espacio de búsqueda. Otra forma del operador de mutación es darle una probabilidad proporcional a la calificación de cada individuo, con el fin de no modificar las mejores soluciones. En otros casos, la mutación se aplica únicamente cuando el resultado genera una mejor solución que la original.

2.5 Computación paralela

Desde su creación, los computadores secuenciales han ido mejorando su velocidad de cómputo de manera constante, pero ahora se empieza a disminuir esta tendencia. Para continuar superando los tiempos pasados se acude entonces al uso de grupos de procesadores para computar datos al mismo tiempo, mejorando la capacidad de procesamiento, sin elevar los costos tanto como otros tipos de desarrollos en este campo. Para una revisión más detallada del tema ver Kumar [1994]; Sterling *et al.* [1995]; Geist [1994]; Vidal y Pérez [2004].

El concepto de computación paralela puede definirse como una división de una tarea entre varios trabajadores que juntos pueden llevarla a cabo en un tiempo menor al empleado por un único trabajador. Así, una tarea específica puede ser completada en menor tiempo si se divide en subtareas que puedan llevarse a cabo simultáneamente por diferentes trabajadores, estableciendo entre ellos una comunicación. Las características más importantes a tener en la cuenta en este tema son: particionamiento de las tareas, comunicación entre tareas o subtareas, identificación clara de los grados de paralelismo a los que se puede llegar según sea el problema, distribución de las actividades y mecanismos de control.

Para un uso eficiente de la computación paralela se deben tener en cuenta [Kumar, 1994] los siguientes aspectos:

Diseño de ordenadores paralelos: pensando en la escalabilidad, rendimiento y soporte de altas velocidades en la comunicación y compartimiento de los datos.

Diseño de algoritmos eficientes: es de poca utilidad un computador en paralelo si no se cuenta con algoritmos paralelos que exploten al máximo su arquitectura.

Métodos para evaluar los algoritmos paralelos: debemos tener una forma de analizar el tiempo de respuesta a problemas y la eficiencia en diferentes escalas.

Lenguajes para ordenadores paralelos: lenguajes lo suficientemente flexibles que permitan una fácil programación e implementaciones eficientes.

Herramientas de programación paralela: alto desarrollo en herramientas para facilitar la comprensión en el momento del desarrollo y la simulación.

Programas paralelos portables: portabilidad, disminución o eliminación de cambios requeridos entre diferentes arquitecturas.

Programación automática de ordenadores paralelos: generación de compiladores que paralelicen un código que no haya sido escrito en paralelo explícitamente.

2.5.1 Modelos de ordenadores paralelos

Tradicionalmente los ordenadores secuenciales se basan en la arquitectura de John Von Neumann, donde dicha estructura se conforma por una unidad central de control (CPU) y una memoria. Los ordenadores que adoptan dicha formación se conocen como *single instruction stream, single data stream* (SISD). La velocidad de dichos computadores tiene dos factores limitantes: la tasa de ejecución de instrucciones y la velocidad a la cual la información se intercambia entre la memoria y la CPU. La segunda se puede mejorar mediante el aumento del número de canales a través de los cuales los datos son accedidos simultáneamente, que se logra mediante el uso de diferentes bancos de memoria que pueden accederse al mismo tiempo. Otra forma de mejorarla es utilizar memorias caché, las cuales son memorias pequeñas, pero de alta velocidad, que sirven como buffer a la memoria principal. Para mejorar la tasa de ejecución de instrucciones se utiliza la ejecución en pipeline, donde se pueden ejecutar instrucciones simultáneamente en diferentes unidades, sobrelapando la búsqueda de una instrucción con la ejecución de la anterior. Sin embargo estas mejoras también tienen sus límites, y el hardware se hace más costoso para mejoras muy pequeñas, por lo que el uso de clusters de computadores para trabajar en paralelo se hace cada vez más popular.

Taxonomía de computadores paralelos

Los computadores paralelos pueden ser construidos de muchas maneras, dependiendo de los mecanismos de control, la organización de espacios de direcciones, la red de interconexión y la granularidad de procesadores.

Mecanismos de control: Las unidades de procesamiento en computadores paralelos pueden operar bajo el control centralizado de una unidad de control o bien

trabajar de forma independiente. Así, se pueden dividir en dos grupos de arquitecturas. La primera (figura 2.8-a) son los computadores *single instruction stream, multiple data stream* (SIMD), donde una unidad de control envía instrucciones a cada unidad de procesamiento. Aquí, una misma instrucción es ejecutada sincrónicamente por todas las unidades de procesamiento. La segunda (figura 2.8-b) son los computadores *multiple instruction stream, multiple data stream* (MIMD), donde cada procesador es capaz de ejecutar un programa diferente.

Los computadores SIMD requieren menos hardware y menos memoria, y son útiles para programas en donde se requiere aplicar un mismo conjunto de instrucciones a una gran cantidad de datos. En un computador MIMD cada procesador tiene su propia unidad de control, pero pueden ser construidos mediante arreglos de computadores de uso general. Así, los computadores MIMD pueden llegar a ser más rápidos y baratos que los SIMD, aunque necesitan mejores algoritmos de sincronización.

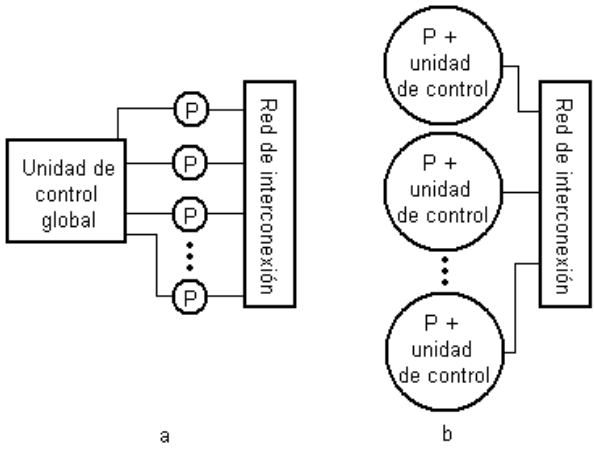


Figura 2.8: Arquitecturas SIMD (a) y MIMD (b)

Redes de interconexión: Las redes de interconexión pueden ser estáticas o dinámicas. En las estáticas o directas la comunicación se da punto a punto con vínculos de comunicación o enlaces (*links*) entre los procesadores como redes de estrella, anillo, árbol o hipercubos. Las redes dinámicas o indirectas son construidas usando switches y enlaces de comunicación para establecer diferentes rutas entre los procesadores, como las redes basadas en bus, multiestado, o tipo crossbar.

Espacio de direcciones: La programación en paralelo requiere de interacción entre los procesadores, y ésta se puede lograr mediante arquitecturas de *paso de mensajes* o de *espacio de direcciones compartido*. La arquitectura de paso de mensajes tiene que ver con sistemas donde los procesadores están conectados usando una red de interconexión y cada procesador tiene una memoria local (privada), y los procesos

de intercambio de datos se realizan por medio de paso de mensajes. A este conjunto de elementos se le suele denominar como multicomputadores. Los computadores con arquitectura de espacio de memoria compartida, tienen un hardware que soporta accesos de lectura y escritura de todos los procesadores a los mismos espacios de esta memoria, aunque en ocasiones se implementa con memorias locales que actúan como caché. El intercambio de información se da al modificar los registros que pueden ser leídos por todos los procesadores. Estos equipos se denominan multiprocesadores.

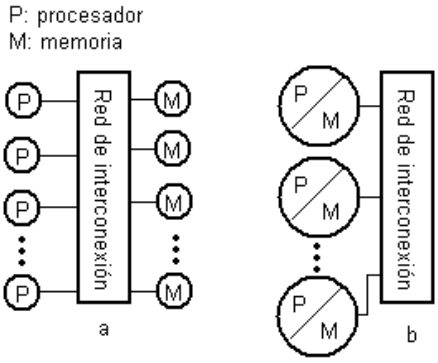


Figura 2.9: Arquitecturas SIMD (a) y MIMD (b)

Granularidad: Los computadores paralelos pueden estar compuestos por pocos procesadores muy potentes, o muchos procesadores de menor capacidad. Los procesos de comunicación y sincronización en la computación paralela crea sobrecargas que disminuyen las prestaciones. La granularidad se define como el tiempo que se utiliza en operaciones de comunicación, sobre el tiempo utilizado en la computación de los datos. Así podemos hablar de grano fino, tarea repartida entre muchos procesadores con un menor tamaño de cada proceso y un mayor número de operaciones de comunicación; o podemos hablar de grano grueso, tarea repartida entre pocos procesadores con un mayor tamaño de cada proceso y un menor número de operaciones de comunicación.

Modelo de un computador paralelo

En un computador paralelo PRAM (parallel random access machine), con memoria compartida y múltiples procesadores, donde hay un único reloj pero los procesadores pueden ejecutar direcciones diferentes al mismo tiempo, hay que manejar de alguna manera las lecturas y escrituras en memoria.

PRAM de lectura exclusiva, escritura exclusiva (EREW): El acceso a una posición de memoria es exclusivo. Es el modelo más débil, permitiendo el mínimo de concurrencia en acceso a memoria.

PRAM de lectura concurrente, escritura exclusiva (CREW): Múltiples lecturas a una posición de memoria son permitidas. Accesos múltiples de escritura son serializados.

PRAM de lectura exclusiva, escritura concurrente (ERCW): Múltiples escrituras a una posición de memoria son permitidas. Accesos múltiples de lectura son serializados.

PRAM de lectura concurrente, escritura concurrente (CRCW): Permite múltiples accesos de lectura y escritura a una posición de memoria.

Permitir lecturas concurrentes no crea discrepancias semánticas en el programa. Sin embargo, accesos de escritura concurrentes requieren arbitración. Algunos protocolos de arbitración son:

- Común: Se permite la escritura concurrente si todos los valores que se intentan escribir son iguales.
- Arbitrario: Un procesador arbitrario puede escribir. El resto fallan.
- Prioridad: Los procesadores tienen un orden de prioridad preestablecido y el de mayor prioridad logra la escritura. El resto fallan.
- Suma: Se escribe el total de la suma de todos los intentos de escritura. (modificable por cualquier operación).

Redes de interconexión dinámicas

Consideremos la implementación de un PRAM EREW como un computador de memoria compartida con p procesadores y m palabras de memoria. Los procesadores se conectan a la memoria mediante switches, que determinan la posición accesada por cada procesador. Para reducir la complejidad del sistema de switches se divide la memoria en bancos.

Redes de conmutación de barra transversal: Una manera simple de conectar p procesadores y b bancos de memoria es mediante un interruptor de barra transversal, que emplea una malla de interruptores o switches. La conexión de un procesador a un banco de memoria no bloquea la conexión de otro. El número de interruptores necesarios es b por cada procesador, por lo cual se hace bastante costoso.

Redes basadas en buses: Los procesadores se conectan a una memoria global por medio de un camino común llamado bus. Cuando un procesador accede a la memoria hace un requerimiento del bus, luego los datos son devueltos por el

mismo bus. Este sistema es simple pero solo permite un número máximo de datos entre la memoria y los procesadores. Si se aumenta el número de procesadores, cada uno gastará más tiempo esperando por el acceso a memoria. Una manera de reducir este cuello de botella es utilizando memorias locales caché en cada procesador.

Redes de interconexión multietapas: Las redes de interconexión multietapas proveen un equilibrio entre costo y rendimiento que no tienen las anteriores. Una conexión común es la red Omega. Esta red contiene $\log(p)$ etapas (p el número de procesadores y bancos de memoria). Cada etapa es un patrón de interconexiones donde existen vínculos entre una entrada i y una salida j si:

$$j = \begin{cases} 2i & \text{, para } 0 < i < \frac{p}{2} - 1 \\ 2i + 1 - p & \text{, para } \frac{p}{2} < i < p - 1 \end{cases}$$

En cada etapa hay $p/2$ interruptores, y cada uno puede actuar como directo o inverso. Las entradas (procesadores) se comunican con las salidas (bancos de memoria) de manera sencilla: en cada etapa el interruptor actúa como directo si los bits en la posición de la etapa en la entrada y la salida son iguales, y como inverso si son diferentes. En algunos casos se puede presentar un bloqueo de comunicación cuando se requiere usar un mismo camino entre 2 etapas.

Redes de interconexión estáticas

Red completamente conectada: Cada procesador tiene comunicación directa con todos los demás. Es un sistema ideal, ya que cada procesador puede enviar mensajes a otro en un solo paso.

Red estrella: Un procesador actúa como el procesador central. Los demás tienen vínculos de conexión con éste. El procesador central es un cuello de botella.

Arreglo lineal y anillo: Es una manera simple de conectar los procesadores, donde cada uno se comunica con otros 2 (excepto los extremos). Cuando se hace una conexión entre los extremos se refiere como anillo. Los procesadores se comunican enviando mensajes hacia la derecha o izquierda y pasando hasta su destinatario.

Malla: Una malla bidimensional es una extensión del arreglo lineal. Cada procesador se comunica directamente con otros 4. Cuando se comunican los procesadores de los extremos se refiere a él como torus. También se puede extender a una malla tridimensional.

Red de árbol: En una red de árbol existe una única ruta entre dos procesadores. En un árbol estático cada nodo es un procesador, mientras que en los dinámicos los nodos intermedios actúan como interruptores y las hojas son procesadores. Para enviar un mensaje el procesador emisor envía el mensaje hacia arriba en el árbol hasta encontrar el procesador o interruptor del sub-árbol más pequeño que contiene al emisor y al receptor. Luego el mensaje es enviado hacia abajo hasta el receptor. Puede haber cuellos de botella en la parte más alta de los mismos, por lo que suelen utilizarse más conexiones a medida que se acerca a la raíz.

Hipercubo: Es una malla multidimensional con 2 procesadores en cada dimensión. Un hipercubo d-dimensional contiene p=2d procesadores. Se puede construir un hipercubo de manera recursiva, así:
- Un hipercubo 0-dimensional es un procesador.
- Un hipercubo 1-dimensional es la conexión de 2 procesadores o hipercubos 0-dimensionales.
- Un hipercubo $(d+1)$-dimensional es construido conectando los procesadores correspondientes de dos hipercubos d-dimensionales.

Dos procesadores están conectados directamente si sus representaciones binarias difieren en un solo bit. Cada procesador de un hipercubo d-dimensional está conectado directamente con d procesadores. El camino con el menor número de conexiones entre dos procesadores está dado por la distancia de Hamming, que consiste en contar el número de bits que difieren en la representación binaria de ambos procesadores. Esta distancia en un hipercubo d-dimensional será de máximo d.

Mecanismos de enrutamiento

Algoritmos eficientes de enrutamiento de mensajes a sus destinos son críticos para el desempeño de un computador paralelo. Un mecanismo de enrutamiento determina el camino que un mensaje sigue por la red desde el emisor hasta el destino. Las entradas son los procesadores fuente y destino, e información del sistema. Las salidas son uno o más caminos posibles. Los algoritmos pueden ser de orden mínimo o no. Los mínimos escogen siempre el camino más corto y pueden provocar congestión. Los no mínimos escogen caminos que pueden ser más largos pero evitan la congestión.

El tiempo que tarda la comunicación entre procesadores es un factor importante en la computación paralela. El tiempo para comunicar un mensaje entre dos procesadores es la latencia, que incluye la preparación del mensaje para el envío y lo que tarda en llegar a su destino. Los principales parámetros para determinar la latencia son el *tiempo de arranque*, que es el tiempo requerido para manejar el mensaje en el

procesador fuente incluyendo el algoritmo de enrutamiento; el *tiempo por salto*, que es el tiempo que tarda un mensaje para pasar de un procesador al siguiente; y el *tiempo de transferencia por palabra*, que es el tiempo que toma una palabra para atravesar el vínculo.

2.5.2 Rendimiento y escalabilidad de sistemas paralelos

Un factor importante en un sistema paralelo, es la medición de los niveles de uso de los recursos y la posibilidad que se tiene de aumentar el trabajo, buscando mejorar el desempeño, por lo que es de gran importancia comprender y analizar una serie de parámetros generales que se dan en el momento de la evaluación de los resultados, tiempos y eficiencias. Los algoritmos secuenciales se evalúan según el tiempo de ejecución en función del tamaño de la salida, los algoritmos paralelos no solo dependen de este factor sino también de su arquitectura y del número de procesadores. Algunos de los parámetros que nos ayudan en el análisis sobre determinadas implementaciones, son los siguientes:

Tiempo de Ejecución: a nivel secuencial, es el tiempo transcurrido entre el inicio y el final de la ejecución (T_s). En un ambiente paralelo es denominado el run-time y consiste en el tiempo que transcurre desde el momento en que el ordenador paralelo comienza, hasta el momento en que el último procesador finaliza la ejecución (T_p).

Aceleración (SpeedUp): es una medida que captura el beneficio relativo de resolver un problema en paralelo, se denomina como S_p y se calcula como T_s/T_p.

Eficiencia: en la teoría la eficiencia de tener p-procesadores es p, pero en la práctica es difícil porque no se puede obtener el 100% de la productividad para computar un algoritmo. La eficiencia es una medida de la fracción de tiempo para la cual un procesador se mantiene en uso, definido a su vez como una proporción entre la velocidad y el número de procesadores. Se denota con E_p y se calcula como S_p/p. Normalmente la representamos como:

$$E_p = \frac{100 \cdot T_s}{T_p \cdot p}$$

Costo: es el producto entre el tiempo de ejecución en paralelo y el número de procesadores usados. El costo también refleja la suma del tiempo que cada procesador gasta resolviendo un problema. La ley de Amdahl dice que la aceleración de un programa paralelo se encuentra limitado por la fracción de tiempo que se consume en realizar operaciones que no se puedan ejecutar en paralelo, por lo cual la aceleración también se encuentra limitada por el tiempo necesario para realizar la comunicación correspondiente a los datos. $T_p = T_s/p + T_c$.

Características para la evaluación de problemas paralelos

Descenso de la escalabilidad: se da cuando se usan menos procesadores que el máximo número posible de ellos al ejecutar un algoritmo paralelo. Un sistema se determina como escalable, si al incrementar el número de procesadores y el tamaño del problema la eficiencia se mantiene igual.

Tamaño del problema: consiste en el número básico de pasos computacionales que se logra en el mejor algoritmo secuencial para resolver un problema sobre un simple procesador. En general se da como el orden del problema a nivel secuencial denotado como W y es equivalente al T_s.

Función de sobrecarga: dicha función en un sistema paralelo se da como parte del costo (tiempo de producción del procesador) que no es incurrido por el algoritmo serial más rápido sobre un ordenador secuencial. (Diferencia entre costo y "run-time" serial). Se calcula como $T_0(W, p) = p \cdot T_p - W$.

Grado de concurrencia: máximo número de tareas que pueden ser ejecutadas simultáneamente en un algoritmo paralelo. Se denota como $C(W)$ siendo W el tamaño del problema, e indica que no más de esta cantidad de procesadores puede ser empleada efectivamente.

Fuentes de sobrecarga paralela: se ve caracterizado por algunos aspectos como la comunicación interprocesador, las cargas desbalanceadas, la extra computación y la distribución de recursos: disco, memoria, entre otros.

Criterios de sobrecarga para tener en cuenta

- Comunicación entre procesadores: cada procesador gasta un tiempo T_c efectuando las comunicaciones, por lo que la comunicación entre procesadores contribuye un tiempo $T_c \cdot p$ a la función de sobrecarga.
- Desbalanceo de cargas: problemas para conocer el tamaño de las subtareas asignadas a varios procesadores, generando un desequilibrio en las cargas que obliga a que algunos procesadores pasen a estados de inactividad, siendo este tiempo un contribuyente más a la función de sobrecarga. Es entonces $(p-1)W_s$ el contribuyente a dicha sobrecarga, donde W_s representa los componentes secuenciales del algoritmo.
- Extra computación: el algoritmo secuencial más rápido conocido puede ser difícil de paralelizar, por lo que se dan algoritmos menos buenos. (alto grado de concurrencia). Entonces $\omega_0 - \omega$ es la contribución a la sobrecarga de trabajo extra que se realiza, donde ω es el tiempo de ejecución del algoritmo más rápido y ω_0 el menos bueno.

2.5.3 Entornos paralelos

Para la computación paralela se debe contar con las herramientas necesarias que nos permitan optimizar realmente los algoritmos y nos ayuden en la búsqueda de soluciones a problemas de gran magnitud. Una forma relativamente barata de hacerlo es mediante la construcción de un cluster de computadoras, utilizando un sistema operativo Linux con entornos MPI o PVM, todo ésto de libre distribución.

La llegada de la computación clusterizada siguiendo el estilo Beowulf, [Sterling *et al.*, 1995] extiende la utilidad de Linux al mundo de la computación paralela de alto rendimiento. Hoy en día, estos valiosos clusters basados en PCs están apareciendo en laboratorios de investigación, departamentos de I+D de empresas, universidades e incluso pequeñas facultades. Si un problema informático no puede solucionarse en un entorno de memoria distribuida flojamente acoplada, un cluster Beowulf puede ser la respuesta, a un precio que los fabricantes tradicionales de computadoras paralelas no pueden llegar.

Construcción de un Beowulf: Cualquiera puede construir un computador paralelo adecuado para la enseñanza de la programación paralela y la ejecución de programas paralelos, muchas veces usando PCs sobrantes o ya existentes. Las PCs en un laboratorio de informática pueden adaptarse para un uso dual gracias a sistemas de encendido dual que les permiten ser encendidas en Linux o Windows, dependiendo de las necesidades del momento. Alternativamente, los equipos no utilizados pueden ser recogidos y fusionados.

Nunca dos clusters Beowulf son iguales. De hecho, sus configuraciones hardware y software son tan flexibles y adaptables que presentan un amplio abanico de posibilidades. Aunque cada cluster es diferente y las configuraciones están dictadas por las necesidades de la aplicación, es posible especificar algunos requerimientos mínimos.

Un nodo tendría que contener, como mínimo, una CPU 486 y una placa madre Intel. Los procesadores Intel 386 funcionarían, pero su rendimiento raramente valdría la pena. Los requerimientos de memoria dependen de los requerimientos de datos de la aplicación y del paralelismo, pero un nodo tendría que contener como mínimo 16MB de memoria. La mayoría de aplicaciones necesitaran 32MB o más por nodo. Usando un espacio de disco centralizado, los nodos se pueden inicializar desde un disquete, un pequeño disco duro o un sistema de archivos en red. Entonces los nodos pueden acceder a su partición raíz desde un servidor de archivos a través de la red, normalmente usando el Network File System (NFS). Esta configuración funciona mejor en un entorno con mucho ancho de banda en las conexiones y con un gran rendimiento en el servidor de archivos. Para un mejor rendimiento bajo otras condiciones, cada nodo tendría que tener suficiente espacio en el disco local para el sistema operativo, la memoria virtual y los datos. Cada nodo tendría que tener como mínimo 200 MB de espacio de disco para los componentes del sistema operativo y la memoria

virtual, pero 400 MB o más permite tener espacio libre que puede ser usado para las aplicaciones en tiempo de ejecución. Como mínimo cada nodo tiene que incluir una tarjeta de red Ethernet o Fast Ethernet. Alternativamente, interconexiones de más alto rendimiento, incluyendo la Gigabit Ethernet y la Myrinet, podrían ser usadas en conjunción con CPUs más rápidas. Finalmente, cualquier tarjeta de video, una lectora de disquetes, la caja y la batería, completan un nodo funcional. Los teclados y los monitores solo se necesitan para la carga y configuración inicial del sistema operativo, a no ser que las máquinas individuales sean usadas interactivamente además de servir como nodos en el sistema paralelo.

Si es posible, los nodos tendrían que estar aislados en una red privada de área local con su propio hub o switch Ethernet. Esto evitará que el tráfico de la red normal interfiera en la comunicación entre los nodos y viceversa. Para incrementar aún más el ancho de banda entre los nodos, se pueden instalar tarjetas de red adicionales en los nodos. Hay disponibles para Linux tanto compiladores comerciales como gratuitos. En general desarrollar código paralelo requerirá un paso explícito de mensajes entre los procesadores utilizando la PVM (*Parallel Virtual Machine* - Máquina virtual paralela), MPI (*Message Passing Interface* - Interfaz de paso de mensajes), u otras librerías de comunicaciones. Ambas, la PVM y la MPI están disponibles gratuitamente y permiten al programador definir fácilmente los nodos usados para ejecutar el código paralelo y pasar datos entre los nodos durante la ejecución usando simples invocaciones a la librería.

Uno de los errores más peligrosos es convertir un problema computacional en un problema de comunicaciones. Esto puede suceder cuando el problema está demasiado dividido, con lo que el tiempo que se necesita para comunicar los datos entre los nodos y sincronizarlos sobrepasa el tiempo de computación real de la CPU. En este caso, usar menos nodos puede resultar en un mejor tiempo de ejecución y una utilización más eficiente de los recursos. Este equilibro entre la carga de computación local en un nodo y las comunicaciones para coordinar esfuerzos entre los nodos tiene que ser optimizado para cada aplicación paralela.

Finalmente, la heterogeneidad de los clusters juega un papel importante en el desarrollo de los algoritmos paralelos. Un factor de dos o más entre las velocidades de las CPUs de los nodos es muy significativo cuando se ejecutan aplicaciones paralelas. Si el trabajo se distribuye uniformemente entre todas las CPUs de un cluster heterogéneo, las CPUs más rápidas tendrán que esperar a las más lentas para completar su parte dentro de la tarea global. Los algoritmos diseñados correctamente pueden superar esta heterogeneidad sobre dividiendo la tarea de forma que se puedan asignar nuevas subtareas a los nodos que completen las anteriores subtareas asignadas.

La computación paralela se utiliza comúnmente en problemas donde se manipulan una gran cantidad de datos. De acuerdo con el alcance inicial planteado en el presente proyecto, se pretendía realizar experimentación con problemas que dieran lugar al uso de la computación paralela. Sin embargo, dadas las condiciones específicas de las empresas que facilitaron su información para nuestras pruebas, el uso de la computación paralela no fue necesario, ya que el número de pedidos en general es pequeño, y por tanto los algoritmos se ejecutan en un tiempo muy corto.

Capítulo 3

Métodos de optimización

En la búsqueda de soluciones a los problemas que enfrenta el hombre cada día, la perfección es algo inherente al proceso y a la solución misma. La meta de optimalidad y eficiencia inspira a científicos, matemáticos y a todas las personas. Desde el surgimiento de los computadores, los métodos de optimización han ido cobrando mayor importancia al permitir la manipulación de grandes cantidades de datos, y así permitir dar solución a problemas que anteriormente podrían ser considerados intratables. La evolución en el campo de la optimización matemática ha sido bastante grande en los últimos cincuenta años. Por consiguiente, cada vez aparecen nuevos retos y nuevos desafíos, gracias a la evolución de los computadores mismos. Los métodos de optimización son fundamentales en la ingeniería y planeación modernas, ya que apoyan los complejos procesos de toma de decisiones. La optimización es básica para cualquier problema que implique toma de decisiones, donde se debe elegir entre diferentes opciones para obtener la mejor alternativa. La optimalidad es el ideal que se persigue, y de acuerdo al contexto puede ser vista de diferentes maneras. Por ejemplo, en el área de la computación se busca optimizar los algoritmos mediante la reducción de los tiempos de ejecución. En el campo de la gerencia de proyectos, el gerente responsable busca optimizar la rentabilidad del proyecto mediante la reducción de costos y tiempos, a la vez que se maximiza el retorno de la inversión. En nuestro caso se habla de optimización matemática, donde se pretende hallar una solución óptima a un problema expresado en términos matemáticos donde existen diferentes alternativas o soluciones posibles. En general, para establecer qué tan buena es cada una de las alternativas se utiliza una función objetivo, que se debe maximizar o minimizar, según sea el caso. Además, las soluciones deben estar contenidas en un cierto dominio, de acuerdo con un conjunto de restricciones propias del problema. La optimización como área de investigación y aplicación ha recibido una gran atención en los últimos años gracias al rápido progreso de la tecnología en la computación.

Para poder llegar a una decisión óptima es necesario contar con información completa sobre el conjunto de todas las posibles decisiones y el beneficio asociado a cada una de ellas. Con esta información podemos elegir la decisión que tenga el mayor beneficio o menor costo, según sea el caso. La calidad de la solución obtenida depende en primera instancia de un buen planteamiento del problema, de su formulación matemática y del proceso de modelación en general. En este proceso se establecen los diferentes elementos que conforman la información que da soporte a la toma de decisiones. Con lo anterior, podemos decir que la representación matemática es una abstracción del problema real que se pretende resolver, pero no siempre es fácil evaluar cuál decisión es mejor que otra, puesto que pueden existir diferentes criterios para tener en cuenta en el costo que se le otorga a cada decisión y algunos pueden incluso ser contrarios. Este tipo de situaciones son comunes, incluso en la vida cotidiana, por ejemplo, al decidir comprar un vehículo entre varias opciones; se tienen diferentes criterios que pueden afectar la decisión, como el precio, la potencia, el consumo de gasolina, el tamaño y muchos más. Para tomar la mejor decisión se tendría que asignar un peso o un costo a cada opción, de acuerdo con la importancia que tenga cada uno de estos criterios para la persona que toma la decisión. Con lo anterior podemos ver que la elección de una función objetivo que refleje realmente el criterio de decisión deseado es de suma importancia, ya que la calidad de la solución obtenida depende de la calidad en la formulación de la función objetivo. Asimismo, es muy importante al plantear el problema saber elegir correctamente las restricciones con el fin de no obtener soluciones que no sean lo que estamos buscando.

3.1 Nociones básicas

Consideremos el problema de optimización:

$$\begin{gathered}\text{minimizar } f(\mathbf{x}) \\ \text{sujeto a } \mathbf{x} \in \Omega\end{gathered}$$

La función $f : \Re^n \to \Re$ que se desea minimizar es una función real-valuada y es llamada la *función objetivo* o *función de costo*. El vector $\mathbf{x}$ es un vector de n variables independientes, es decir, $\mathbf{x} = [x_1, x_2, ..., x_n]^T \in \Re^n$. Las variables $x_1, ..., x_n$ son llamadas *variables de decisión*. El conjunto Ω es un subconjunto de $\Re^n$ llamado el conjunto de *restricciones* o conjunto de *soluciones factibles*.

Este problema de optimización puede ser visto como un problema de decisión que implica encontrar el *mejor* vector $\mathbf{x}$ de variables de decisión entre todos los posibles vectores en Ω. Por el mejor se puede entender el vector que resulte en el menor valor de la función objetivo. En caso de que existan varios vectores con el mismo valor mínimo, encontrar uno de ellos será suficiente. Existen también problemas de optimización donde interesa maximizar la función objetivo, pero éstos pueden ser

representados de la misma forma, ya que maximizar f es equivalente a minimizar $-f$. El problema que planteamos es un problema general de optimización con restricciones, en el caso de que no existan restricciones se puede simplemente asumir $\Omega = \Re^n$.

Para tener una visión gráfica del problema supongamos una función $f : \Re^2 \to \Re$ con las restricciones $x_1 + x_2 \leq 40$, $2x_1 + x_2 \leq 60$ y $x_1, x_2 \geq 0$. Podemos dibujar el conjunto de puntos que satisfacen estas restricciones como se aprecia en la Figura 3.1. En este caso la región factible corresponde al área coloreada en gris y la solución óptima se hallaría sobre el perímetro de dicha área, dependiendo de cuál sea la función objetivo $f(x_1, x_2)$.

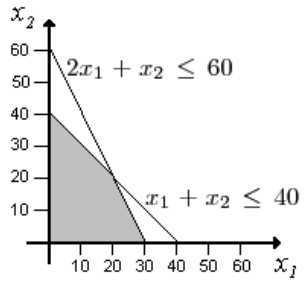


Figura 3.1: Región factible para un problema con dos variables

Supongamos la función objetivo $f(x_1, x_2) = 2x_1 - x_2 + 10$. Podemos entonces plantear este problema matemáticamente de la siguiente forma:

$$\max z = 2x_1 - x_2 + 10$$

Sujeto a

$$\begin{aligned} x_1 + x_2 &\leq 40 \\ 2x_1 + x_2 &\leq 60 \\ x_1 &\geq 0 \\ x_2 &\geq 0 \end{aligned}$$

Si evaluamos la función objetivo en los vértices de la región factible como se muestra a continuación, es fácil ver que la solución óptima, es decir el máximo de la función objetivo, se encuentra en el punto $x_1 = 30$, $x_2 = 0$ y tiene un valor máximo de $z = 70$.

x_1	x_2	z
0	0	10
0	40	-30
30	0	**70**
20	20	30

En el campo de la optimización matemática existen diversos problemas clásicos que han sido tratados de diferentes formas y que poseen gran importancia debido a la diversidad de situaciones reales en que pueden ser utilizados. A continuación se describen algunos de estos problemas clásicos de optimización.

3.2 Problemas clásicos

En la literatura de los métodos de optimización aparecen diferentes problemas típicos a partir de los cuales se pueden formular una gran cantidad de problemas reales y para los cuales existen ya algunos métodos de solución. En esta sección se presentan algunos de los métodos más comunes que agrupan a su vez una gran variedad de problemas específicos. Los ejemplos dados en la mayoría de problemas son basados en los ejemplos de Winston [2005].

3.2.1 El problema de transporte

En general, un problema de transporte se especifica por la información siguiente:

1. Un conjunto de m *puntos de suministro* a partir de los cuales se envía un bien. El punto de suministro i abastece a lo sumo s_i unidades.

2. Un conjunto de n *puntos de demanda* a los cuales se envía el bien. El punto de demanda j debe recibir por lo menos d_j unidades del producto enviado.

3. Cada unidad producida en el punto de suministro i y enviada al punto de demanda j incurre en un *costo variable* de c_{ij}.

La figura 3.2 muestra una interpretación gráfica del problema. El planteamiento matemático puede darse así: Sea x_{ij} el número de unidades enviadas desde el punto de suministro i al punto de demanda j, entonces la formulación general de un problema de transporte es:

$$\min \sum_{i=1}^{m} \sum_{j=1}^{n} c_{ij} x_{ij}$$

sujeto a

$$\sum_{j=1}^{n} x_{ij} \leq s_i \ (i = 1, 2, ..., m) \text{ (restricciones de capacidad)}$$

$$\sum_{i=1}^{m} x_{ij} \geq d_j \ (j = 1, 2, ..., n) \text{ (restricciones de demanda)}$$

$$x_{ij} \geq 0 \ (i = 1, 2, ..., m; j = 1, 2, ..., n)$$

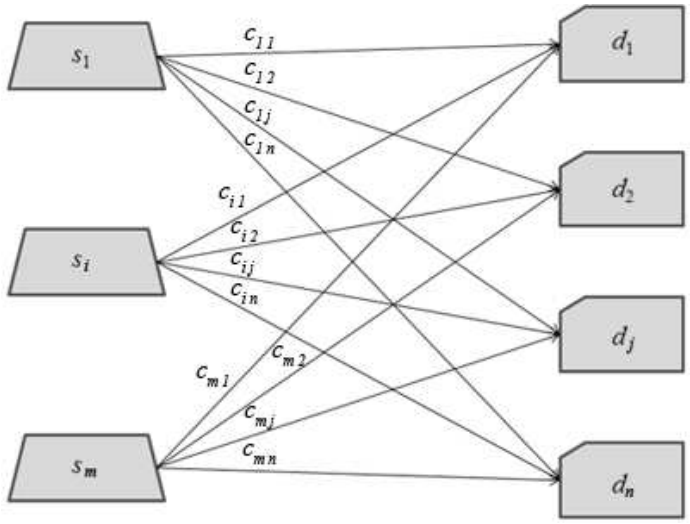


Figura 3.2: Representación gráfica de un problema de transporte.

En general, cuando número de puntos de suministro y de demanda es muy alto, la complejidad del problema crece bastante y se hace prácticamente intratable sin la ayuda de un método de optimización por computador.

Ejemplo

La empresa de refrescos Coco Loco cuenta con tres plantas de producción y cuatro centros de distribución en el país. La producción semanal de las plantas de producción es de 35, 50 y 40 miles de botellas de refresco por semana respectivamente, mientras la demanda en cada centro de distribución es de 45, 20, 30 y 30 miles de botellas de refresco por semana. Los costos de enviar el producto desde una planta de producción hasta un centro de distribución depende de la distancia entre ambos y se puede apreciar en la tabla 3.1.

Tabla 3.1: Capacidad, demanda y costo de envío entre las plantas y centros de distribución de la empresa Coco Loco.

	A centro de distribución				
De	1	2	3	4	Capacidad
Planta 1	\$8	\$6	\$10	\$9	35
Planta 2	\$9	\$12	\$13	\$7	50
Planta 3	\$14	\$9	\$16	\$5	40
Demanda	45	20	30	30	

Solución. Para formular el problema debemos definir las variables de decisión de acuerdo con las decisiones que debe tomar la empresa Coco Loco. En este caso debemos determinar el número de refrescos que se envían desde cada planta a cada centro de distribución. Sea x_{ij} el número (en miles) de botellas de refresco producidos en la planta i y enviados al centro de distribución j. En términos de estas variables de decisión, el costo total de suministrar los refrescos a cada centro de distribución se puede escribir como $C = 8x_{11} + 6x_{12} + 10x_{13} + 9x_{14} + 9x_{21} + 12x_{22} + 13x_{23} + 7x_{24} + 14x_{31} + 9x_{32} + 16x_{33} + 5x_{34}$. Además se cuenta con las siguientes restricciones de capacidad, de acuerdo con la producción máxima de cada una de las plantas semanalmente:

$$x_{11} + x_{12} + x_{13} + x_{14} \leq 35 \text{ (restricción de capacidad de la planta 1)}$$

$$x_{21} + x_{22} + x_{23} + x_{24} \leq 50 \text{ (restricción de capacidad de la planta 2)}$$

$$x_{31} + x_{32} + x_{33} + x_{34} \leq 40 \text{ (restricción de capacidad de la planta 3)}$$

De la misma manera, se tienen las siguientes restricciones de la demanda que se debe satisfacer en cada centro de distribución:

$$x_{11} + x_{21} + x_{31} \geq 45 \text{ (restricción de demanda del centro 1)}$$

$$x_{12} + x_{22} + x_{32} \geq 20 \text{ (restricción de demanda del centro 2)}$$

$$x_{13} + x_{23} + x_{33} \geq 30 \text{ (restricción de demanda del centro 3)}$$

$$x_{14} + x_{24} + x_{34} \geq 30 \text{ (restricción de demanda del centro 4)}$$

Finalmente tenemos la restricción de que cada variable de decisión debe tomar un valor no negativo, así $x_{ij} \geq 0$ para todo i, j. En la figura 3.3 se puede apreciar una representación gráfica de este problema.

La solución óptima para este problema es $C = 1020$ y está dada por $x_{12} = 10$, $x_{13} = 25$, $x_{21} = 45$, $x_{23} = 5$, $x_{32} = 10$, $x_{34} = 30$.

Problemas de asignación

En un *problema de asignación* se tienen m recursos (personas o máquinas) y n tareas por completar. Cada recurso i tarda un cierto tiempo c_{ij} en completar la tarea j. Se debe asignar un recurso a cada tarea de manera que el tiempo total para finalizar todas las tareas sea mínimo. Cada recurso debe asignarse exactamente a una tarea, y a su vez cada tarea debe tener asignado exactamente un recurso. El problema de asignación puede verse entonces como un problema de transporte donde los suministros y demandas son iguales a 1. Los recursos pueden verse como puntos de suministro y las tareas como puntos de demanda. El problema consiste en asignar de una manera óptima los diferentes recursos en un proceso dado. La figura 3.4 ilustra gráficamente este problema.

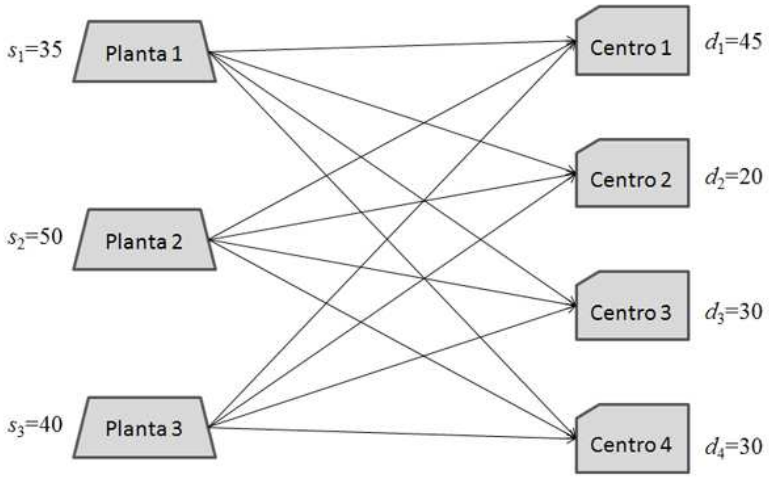


Figura 3.3: Representación gráfica del problema de transporte de la empresa Coco Loco con capacidad y demanda.

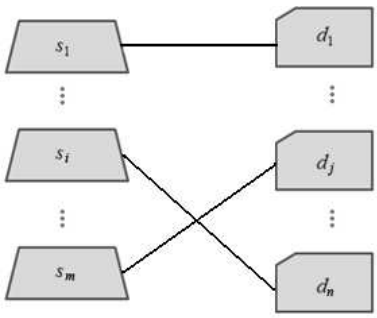


Figura 3.4: Representación gráfica de un problema de asignación.

Problemas de transbordo

Un problema de transporte sólo permite envíos que van directamente de un punto de suministro a un punto de demanda. En muchas situaciones se permiten envíos entre puntos de suministro o entre puntos de demanda. Algunas veces también puede haber puntos por los que se podría hacer el transbordo de los bienes en su viaje de un punto de suministro a un punto de demanda. Un *problema de transbordo* cumple con estas características, es decir, es un problema de transporte que permite el envío entre puntos de suministro y entre puntos de demanda, además de contener puntos de transbordo por los que se podrían enviar los bienes en su trayecto de un punto de suministro a un punto de demanda. La figura 3.5 ilustra este problema.

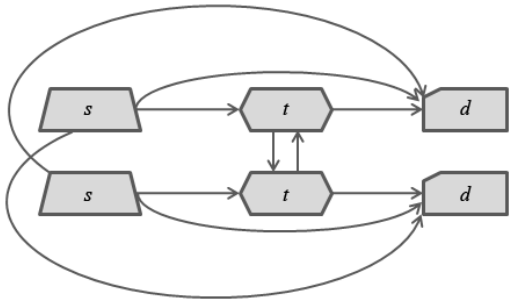


Figura 3.5: Representación gráfica de un problema de transbordo.

Problemas de trayectoria más corta

Una *red* o *grafo* se define mediante un conjunto de *nodos* y un conjunto de *arcos*. Sea V el conjunto de vértices o nodos de la red y sea A el conjunto de arcos. Un arco consiste en un par ordenado de nodos y representa una posible dirección de movimiento que podría ocurrir entre dichos puntos. Si una red contiene el arco (i, j) entonces es posible moverse del nodo i al nodo j. Una secuencia de arcos tal que el nodo terminal de cada arco es idéntico al nodo inicial del siguiente es una *trayectoria*. Por ejemplo, para una red de cuatro nodos, la secuencia $(1, 2) - (2, 3) - (3, 4)$ es una trayectoria que representa una forma de viajar del nodo 1 al nodo 4. El *problema de la trayectoria más corta* consiste en encontrar la trayectoria con menor costo entre un nodo de origen y un nodo destino, pasando por diferentes nodos intermedios, donde cada arco o *trayecto* tiene un costo asociado. Los problemas de trayectoria más corta pueden ser considerados como problemas de transbordo, donde se debe minimizar el costo de enviar una unidad del nodo i al j con los demás nodos de la red como puntos de transbordo. La figura 3.6 ilustra un ejemplo de este tipo de problemas.

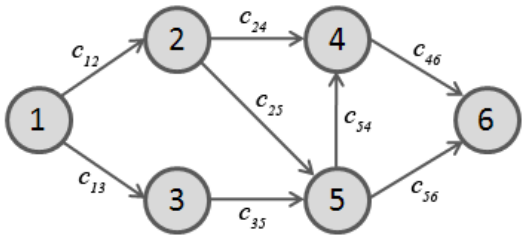


Figura 3.6: Representación gráfica de trayectoria más corta.

3.2.2 El problema de la mochila

El problema de la mochila es un problema típico de la programación entera (*ver sección 3.3.2*), en la cual todas las variables deben pertenecer al conjunto de los enteros. Suponga que hay una excursión para la cual se puede llevar una mochila que, lógicamente, tiene una capacidad limitada. El problema consiste entonces en escoger los objetos que se llevarán en la mochila, de manera que el beneficio obtenido de ellos sea máximo, sin sobrepasar la capacidad de la mochila. Este problema puede ser trasladado a diferentes ámbitos donde se debe elegir entre diferentes opciones con una única restricción de capacidad. Las variables de decisión deben tomar valores enteros en caso que existan copias ilimitadas de cada tipo de objeto, o incluso estar restringidos a 1 y 0, en caso que solo exista un objeto de cada tipo. Este tipo de problema se presenta con cierta frecuencia en los ámbitos económico e industrial, donde la mochila suele representar la restricción de presupuesto y donde la utilidad de los objetos seleccionados se equipara a un beneficio económico por adquirir o llevar a cabo ciertas acciones. Por ejemplo, puede aplicarse al decidir cómo invertir las ganancias de una empresa sobre un portafolio de posibles inversiones. La capacidad de la mochila equivaldría al total de dinero disponible para invertir, y cada posible inversión tendría un costo (es decir, la cantidad que se invierte) y un beneficio esperado o utilidades. También cuando una organización, por ejemplo, un equipo de fútbol, se plantea realizar una nueva contratación, puede analizar los posibles candidatos según el salario de cada uno como el costo en la restricción de presupuesto y el aporte esperado de cada uno de ellos como el beneficio en la función objetivo.

Ejemplo

Juan está planeando un viaje y tiene una maleta con capacidad de 10Kg en la que debe llevar productos para vender. Para llenar la maleta Juan cuenta con numerosos artículos de tres tipos: el producto 1 tiene un peso de 4Kg y un valor de $11, el producto 2 tiene un peso de 3Kg y un valor de $7 y el producto 3 tiene un peso de 5Kg y un valor de $12. ¿Cómo puede Juan aprovechar al máximo la capacidad de su

maleta llevando los productos que mayor valor representen?

Solución. Definimos las variables de decisión de la siguiente manera: sea x_i el número de artículos del producto i que lleva Juan en la mochila. El problema se puede entonces plantear como:

$$\max C = 11x_1 + 7x_2 + 12x_3$$

sujeto a

$$4x_1 + 3x_2 + 5x_3 \leq 10$$

$$x_i \in \mathbb{Z}$$

El beneficio máximo que se puede lograr es $C = \$25$ con $x_1 = 1$, $x_2 = 2$ y $x_3 = 0$, es decir, llevando Juan un artículo del producto A y dos artículos del producto B en su maleta.

3.2.3 El problema del agente viajero

El problema del agente viajero o TSP (de sus siglas en inglés - *Traveling Salesman Problem*) consiste en hallar la mejor ruta para un agente viajero que debe visitar n ciudades de manera que la distancia recorrida (o el costo de recorrer cada trayecto) sea mínimo, visitando cada ciudad exactamente una vez y regresando finalmente al punto de partida. En este caso la decisión que se debe tomar es el orden en que se visita cada una de las ciudades. El espacio de soluciones de este problema crece en forma factorial, es decir, para un problema TSP con 12 ciudades se tienen más de 400 millones de posibles soluciones. Este problema es uno de los más estudiados en el ámbito de optimización combinatoria. Este problema tiene numerosas aplicaciones prácticas, además de las más evidentes en áreas de logística de transporte, donde se deben optimizar rutas de reparto. Por ejemplo, en robótica, permite resolver problemas de fabricación para minimizar el número de desplazamientos al realizar una serie de perforaciones en una plancha o en un circuito impreso. También puede ser utilizado en control y operativa optimizada de semáforos. También en el área de telecomunicaciones puede ser aplicado para decidir rutas de propagación de datos y diseño de redes digitales.

Ejemplo

David es el presidente de una compañía y debe visitar sus agencias en 5 ciudades para hacerle seguimiento a su negocio. El valor de los tiquetes entre cada par de ciudades se puede apreciar en la tabla 3.2. ¿Cómo puede David minimizar el costo del viaje visitando las cinco ciudades y finalizando en el mismo lugar?

Tabla 3.2: Costo de traslado entre las ciudades para el agente viajero.

	Armenia	Bogotá	Cali	Medellín	Pasto
1. Armenia	0	132	217	164	58
2. Bogota	132	0	290	201	79
3. Cali	217	290	0	113	303
4. Medellín	164	201	113	0	196
5. Pasto	58	79	303	196	0

Solución. David debe visitar las cinco ciudades en el orden que le signifique el menor costo. Definamos:

$$x_{ij} = \begin{cases} 1 & \text{si David viaja de la ciudad } i \text{ a la } j \\ 0 & \text{si no sucede así} \end{cases}$$

$c_{ij} =$ costo del traslado de la ciudad i a la j

Podemos plantear el problema de la siguiente manera:

$$\min C = \sum_{i=1}^{5} \sum_{j=1}^{5} c_{ij} x_{ij}$$

sujeto a

$$\sum_{j=1}^{5} x_{ij} = 1 \ (i = 1, 2, ..., 5)$$

$$\sum_{i=1}^{5} x_{ij} = 1 \ (j = 1, 2, ..., 5)$$

$$x_{ij} \in \{0, 1\} \ (i = 1, 2, ..., 5; j = 1, 2, ..., 5)$$

El costo mínimo que se puede lograr es $C = 668$ con $x_{15} = x_{52} = x_{24} = x_{43} = x_{31} = 1$, es decir, realizando David el recorrido Armenia-Pasto-Bogotá-Medellín-Cali-Armenia.

3.2.4 El problema de corte de material

En el caso general del problema de corte bidimensional, se dispone de una superficie S de algún material con dimensiones $L \times W$ determinadas. Además, se dispone de un conjunto de n patrones distintos, cada uno de los cuales tiene unas dimensiones $l_i \times w_i$ y un beneficio b_i asociado. El problema consiste entonces en encontrar la distribución de patrones sobre la superficie S que maximice el beneficio obtenido y minimice el desperdicio de material (i.e., el material sobrante donde no se utiliza ningún patrón). Este problema puede plantearse en diferentes formas y puede darse en tres dimensiones o en una sola, como se explicó en la sección 2.2. Este problema

se aplica en numerosas industrias, como por ejemplo en el corte de telas para hacer prendas de vestir, corte de troncos de madera, corte de láminas metálicas, corte de papel, cartón, entre otros. En la figura 3.7 se aprecian imágenes del proceso de corte de cuero animal para producir carteras, bolsos, zapatos y demás.

Figura 3.7: El proceso de corte del cuero

Figura 3.8: Máquinas de corte

Ejemplo

La empresa Woodstock vende tablas de 3, 5 y 9 metros que recorta a partir de tablones de 17 metros. Para esta semana Woodstock debe entregar a sus clientes pedidos por 25 tablas de 3m, 20 tablas de 5m y 15 tablas de 9m. ¿Cómo se deben recortar los

tablones de 17m de manera que el desperdicio de material sea mínimo?

Solución. Las decisiones que debe tomar Woodstock corresponden al modo de recortar cada uno de los tablones de 17m. Las posibles formas de recortarlos se muestran en la tabla 3.3. No se consideran patrones con más de 3m de desperdicio, pues se podría utilizar dicho desperdicio para recortar tablas útiles.

Tabla 3.3: Desperdicio para los posibles patrones en el problema de corte.

	Tablas de 3m	Tablas de 5m	Tablas de 9m	Desperdicio
1	5	0	0	2
2	4	1	0	0
3	2	2	0	1
4	2	0	1	2
5	1	1	1	0
6	0	3	0	2

Sea x_i el número de tablones recortados con el patrón i. Sabemos que lo que interesa minimizar es el desperdicio de material, mientras que las restricciones consisten en satisfacer los pedidos mínimos de los clientes. Así, podemos plantear el problema de la siguiente manera:

$$\min T = 2x_1 + 0x_2 + x_3 + 2x_4 + 0x_5 + 2x_6$$

sujeto a

$$\begin{aligned} 5x_1 + 4x_2 + 2x_3 + 2x_4 + x_5 &\geq 25 \text{ (restricción de tablas de 3m)} \\ x_2 + 2x_3 + x_5 + 3x_6 &\geq 20 \text{ (restricción de tablas de 5m)} \\ x_4 + x_5 &\geq 15 \text{ (restricción de tablas de 9m)} \\ x_1, x_2, x_3, x_4, x_5 &\geq 0 \end{aligned}$$

Al resolver este problema encontramos que el desperdicio mínimo que se puede lograr es de 2m y se obtiene con $x_2 = 3$, $x_5 = 15$ y $x_6 = 1$, es decir, recortando 3 tablones con el patrón 2, 15 tablones con el patrón 5 y un tablón con el patrón 6. El total de tablas que se obtienen son 27 de 3m, 21 de 5m y 15 de 9m, satisfaciendo plenamente los pedidos de los clientes.

3.3 Algoritmos de optimización

Los problemas de optimización presentados en la sección anterior requieren de algoritmos robustos y rápidos, que proporcionen confiabilidad y agilidad en la búsqueda de una solución al problema. Un factor clave para el éxito de un método de optimización es que permita incorporar a su estrategia de búsqueda el conocimiento disponible de

su propio dominio. Así, para diferentes modelos y problemas existen métodos y algoritmos de optimización que pueden ser más útiles o eficaces que otros bajo diversas condiciones. En esta sección se se muestran algunos de los algoritmos de optimización más comunes.

3.3.1 Aspectos generales

De acuerdo con sus características, los métodos de optimización pueden clasificarse de diferentes formas. En términos generales, podemos distinguir dos grandes grupos: métodos determinísticos y métodos probabilísticos o heurísticos. El primer grupo comprende los métodos que se basan en decisiones determinísticas para llegar a una solución, es decir, alcanzan la solución ejecutando una cantidad finita de operaciones. Los métodos determinísticos son generalmente más fáciles y rápidos de implementar, pero pueden tener en cuenta todos los elementos necesarios y adecuados en los modelos, debido a que en general estos métodos para su solución requieren de una geometría muy simple. Además, debido a que en el proceso de solución cada valor obtenido depende de los anteriores, el error de redondeo puede llegar a ser muy grande, lo que nos llevaría a una solución inadecuada. Los métodos determinísticos no siempre pueden ser aplicados al problema tratado, debido a la gran cantidad de cálculos y recursos computacionales que pueden llegar a requerir. Por otro lado, los métodos probabilísticos, como su nombre lo indica, son aquellos que se basan en probabilidades y variables estocásticas para encontrar una solución aproximada a un problema. Dicha aproximación es por lo general lo suficientemente buena para ser utilizada en los problemas prácticos cotidianos. Es decir, las condiciones propias del problema no plantean exigencias más allá de la calidad de la solución encontrada. Es importante destacar que una herramienta fundamental para la aplicación de estos métodos es el computador, debido a la cantidad de cálculos que se deben realizar.

Las algoritmos heurísticos son técnicas de aproximación que se han utilizado desde los comienzos de la investigación de operaciones para resolver difíciles problemas combinatorios. Con el desarrollo de la teoría de la complejidad a principio de los años 70, se hizo evidente que como la mayoría de estos problemas son de tipo NP-Duros, habría pocas esperanzas de hallar eficientemente una solución exacta para ellos. Este hecho le dio gran importancia a la investigación y desarrollo de métodos heurísticos de solución [Gendreau, 2002]. Los algoritmos más populares son los algoritmos de búsqueda, basados principalmente en la técnica de Búsqueda Local. La búsqueda local es un método iterativo que comienza de una solución factible inicial y la mejora progresivamente aplicando una serie de modificaciones locales. En cada iteración, el algoritmo se mueve a una mejor solución factible que difiere muy poco de la solución anterior (i.e., se encuentra en el vecindario de la solución anterior). El algoritmo termina cuando el mínimo local es hallado.

En 1975, John Holland publica su método de Algoritmos Genéticos, un heurístico basado en la teoría evolutiva de Darwin. En 1983, los autores Kirkpatrick, Gelatt y Vecchi presentan un nuevo heurístico llamado Recocido Simulado (o *Simulated Annealing*), que demostraba convergencia a una solución óptima de un problema combinatorio en un tiempo aceptable de computación. En analogía con la estadística mecánica, el recocido simulado puede interpretarse como una caminata aleatoria controlada sobre el espacio de soluciones. Posteriormente, el interés de la comunidad científica por crear nuevos métodos heurísticos aumentó, y en los años siguientes se proponen nuevas analogías con el mundo natural, tales como la Búsqueda Tabú, Colonias de Hormigas, entre otros. Junto con los métodos anteriores, estos métodos fueron ganando popularidad desde entonces. Hoy en día se les conoce con el nombre de meta-heurísticos, y se han convertido en la base de los métodos heurísticos para resolver problemas de optimización combinatoria [Gendreau, 2002].

En la Figura 3.9 se presenta una clasificación general de los métodos de optimización y a continuación se hace una revisión de las principales características de algunos de ellos.

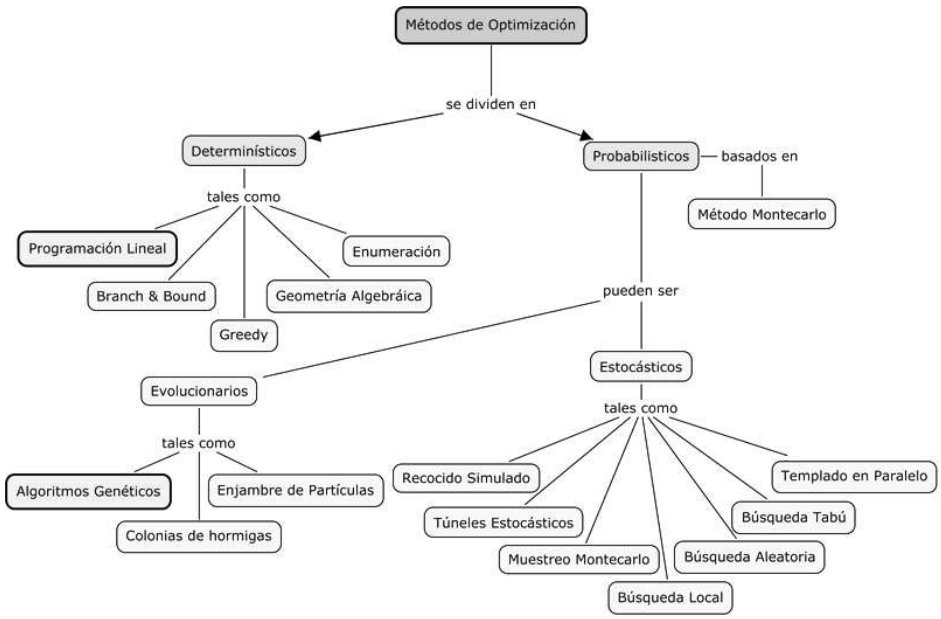


Figura 3.9: Métodos de optimización

3.3.2 Programación lineal

La programación lineal se presenta con mayor detalle en la sección 2.3. Es un método de optimización para problemas donde tanto la función objetivo como las restric-

ciones del problema pueden ser expresadas en forma de funciones lineales. Según Winston [2005], un problema de programación lineal es un problema de optimización que cumple lo siguiente:

1. Se intenta maximizar (o minimizar) una función lineal de las variables de decisión. La función que se desea maximizar (o minimizar) se llama función objetivo.

2. Los valores de las variables de decisión deben satisfacer un conjunto de restricciones. Cada restricción debe ser una ecuación lineal o una desigualdad lineal.

3. Se relaciona una restricción de signo con cada variable. Para cualquier variable x_i, la restricción de signo especifica que x_i no debe ser negativa.

4. Hay certeza sobre todos los parámetros, es decir que se conoce con certeza el valor de todos los coeficientes o la contribución de todas las variables de decisión sobre la función objetivo y las restricciones.

El algoritmo Símplex presentado en la sección 2.3 es un método determinístico para resolver problemas planteados como problemas de programación lineal. Este algoritmo ejecuta una serie de pasos fijos con los cuales se encuentra la solución óptima global. Con este método siempre se encuentra la solución óptima al problema, si esta existe.

La programación lineal es ampliamente usada en diferentes problemas cotidianos que pueden ser planteados fácilmente de esta manera. Sin embargo, la gran mayoría de los problemas que se presentan en las organizaciones tienen una complejidad mayor, haciendo difícil plantearlos mediante funciones lineales, y por tanto este método pierde su utilidad.

Programación entera

En términos generales, un problema de programación entera es un problema de programación lineal en el cual se requiere que algunas o todas las variables sean enteros no negativos y son normalmente más difíciles de resolver que un problema de programación lineal normal. Un problema de programación entera en el cual se requiere que todas las variables sean enteros se denomina un problema puro de programación con enteros [Winston, 2005].

Definición (Relajación del PL). El problema de programación lineal obtenido cuando se omiten todas las restricciones de enteros para las variables en un problema de programación entera se llama *relajación del PL* de la programación entera.

Cualquier problema de programación entera puede considerarse como la relajación del PL más las restricciones adicionales que establecen cuáles variables deben ser enteros. Por tanto, la relajación del PL es una versión de la programación entera

con menos restricciones, es decir, relajada. Además, la región factible para cualquier problema de programación entera debe estar contenida en la región factible para la relajación del PL correspondiente.

Ejemplo. Supongamos el siguiente problema de programación entera:

$$\max z = 21x_1 + 11x_2$$

sujeto a

$$\begin{aligned} 7x_1 + 4x_2 &\leq 13 \\ x_1, x_2 &\geq 0 \\ x_1, x_2 &\in \mathbb{Z} \end{aligned}$$

Como se aprecia en la figura 3.10, la región factible para este problema consiste en el conjunto de puntos $S = \{(0,0), (0,1), (0,2), (0,3), (1,0), (1,1)\}$. Si se calculan y se comparan los valores de z para cada uno de los seis puntos en la región factible, se encuentra que la solución óptima es $z = 33$, con $x_1 = 0$ y $x_2 = 3$.

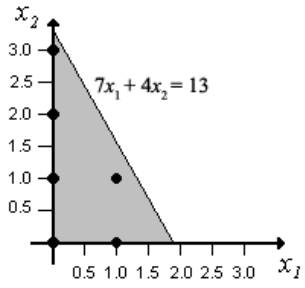


Figura 3.10: Región factible para el problema de PE y la relajación del PL.

En el caso que la región factible del problema de relajación del PL sea acotada, tendríamos un número finito de puntos en la región factible del problema de programación entera. Para encontrar la solución óptima en este conjunto, se podría hacer la enumeración de todas las posibles soluciones y evaluar la función objetivo con cada una de ellas. Sin embargo, para problemas reales con millones de posibles soluciones, este método claramente no es el mejor. Otra práctica común es resolver el problema de relajación del PL y posteriormente hacer un redondeo de las variables y así cumplir con la restricción de los enteros. Lastimosamente, esta tampoco es una buena opción, ya que las variables redondeadas podrían estar por fuera de la región factible o, aunque estén dentro de la región, no ser la solución óptima al problema. Para mayor información sobre la correcta solución a problemas de programación lineal el lector puede remitirse a Winston [2005].

3.3.3 Programación no lineal

En la programación lineal se analizan problemas donde el objetivo es optimizar una función lineal sujeta a restricciones lineales. Sin embargo, en muchos de los problemas comunes de optimización, la función objetivo es una función no lineal, o bien, algunas de las restricciones son funciones no lineales. A este tipo de problemas de optimización se les llama problemas de *programación no lineal* (PNL). Un problema de programación no lineal general [Winston, 2005] se expresa como sigue:

$$\max z = f(x_1, x_2, \ldots, x_n)$$

sujeto a

$$\begin{aligned} g_1(x_1, x_2, \ldots, x_n) &\leq b_1 \\ g_2(x_1, x_2, \ldots, x_n) &\leq b_2 \\ \vdots &\quad \vdots \\ g_m(x_1, x_2, \ldots, x_n) &\leq b_m \end{aligned}$$

La región factible para el PNL es el conjunto de puntos $(x_1, x_2, \ldots, x_n)$ que satisfacen las m restricciones del problema. Cualquier punto $\bar{x}$ en la región factible para el cual $f(\bar{x}) \geq f(x)$ para todos los puntos x en la región factibles una solución óptima del problema.

Ejemplo

Si se utilizan K unidades de materia prima y L unidades de trabajo, una compañía puede producir KL unidades de un producto dado. La materia prima se puede comprar a \$4/unidad y la mano de obra a \$1/unidad. Se dispone de un total de \$8 para comprar materia prima y mano de obra. ¿De qué manera puede la empresa maximizar la cantidad del producto fabricado?

Sea K las unidades de materia prima compradas y L las unidades de mano de obra contratadas. Entonces el problema se puede plantear de la siguiente manera:

$$\max z = KL$$

sujeto a

$$\begin{aligned} 4K + L &\leq 8 \\ K, L &\geq 0 \end{aligned}$$

Es fácil ver que la función $z = KL$ es una función no lineal. La solución óptima a este problema se encuentra en $K = 1$, $L = 4$ y toma un valor máximo de $z = 4$.

3.3.4 Programación dinámica

Existe una serie de problemas cuyas soluciones pueden ser expresadas recursivamente en términos matemáticos, y posiblemente la manera más natural de resolverlos es mediante un algoritmo recursivo. Sin embargo, el tiempo de ejecución de la solución recursiva, normalmente de orden exponencial y por tanto impracticable, puede mejorarse substancialmente mediante la *programación dinámica* [Guerequeta y Vallecillo, 2000]. En el diseño divide y vencerás, para resolver un problema, se debe dividir en subproblemas independientes, los cuales se resuelven de manera recursiva para combinar finalmente las soluciones y así resolver el problema original. El inconveniente se presenta cuando los subproblemas obtenidos no son independientes sino que existe solapamiento entre ellos; entonces es cuando una solución recursiva no resulta eficiente por la repetición de cálculos que conlleva. En estos casos es cuando la programación dinámica nos puede ofrecer una solución aceptable. La eficiencia de esta técnica consiste en resolver los subproblemas una sola vez, guardando sus soluciones en una tabla para su futura utilización. La Programación Dinámica no sólo tiene sentido aplicarla por razones de eficiencia, sino porque además presenta un método capaz de resolver de manera eficiente problemas cuya solución ha sido abordada por otras técnicas y han fracasado. La solución de problemas mediante esta técnica se basa en el llamado principio de óptimo enunciado por Bellman en 1957 y que dice: "*En una secuencia de decisiones óptima toda subsecuencia ha de ser también óptima*".

La programación dinámica se utiliza para resolver diversos problemas de optimización. Por lo general, esta técnica llega a la solución trabajando hacia atrás, partiendo del final del problema hacia el principio, por lo que un problema enorme e inmanejable se convierte en una serie de problemas más pequeños y manejables. Los problemas de programación dinámica cumplen en general con las siguientes características [Winston, 2005]:

- Es posible dividir el problema en etapas, y se requiere de una decisión en cada etapa.
- Cada etapa se relaciona con una cierta cantidad de etapas, es decir, las decisiones que se toman en una etapa dependen del estado obtenido en otras etapas.
- La decisión tomada en cualquier etapa describe el modo como el estado en la etapa actual se transforma en el estado de la etapa siguiente.
- Dado el estado actual, la solución óptima para cada una de las etapas restantes no tiene que depender de los estados ya alcanzados o de las decisiones previas.
- Si los estados del problema se clasifican dentro de uno de T etapas, debe haber una recursión que relacione el costo o la recompensa ganada durante las etapas

$t, t+1, \ldots, T$ con el costo o la recompensa ganada a partir de las etapas $t+1, t+2, \ldots, T$.

En grandes líneas, el diseño de un algoritmo de programación dinámica consta de los siguientes pasos:

1. Planteamiento de la solución como una sucesión de decisiones y verificación de que ésta cumple el principio de óptimo.
2. Definición recursiva de la solución.
3. Cálculo del valor de la solución óptima mediante una tabla en donde se almacenan soluciones a problemas parciales para reutilizar los cálculos.
4. Construcción de la solución óptima haciendo uso de la información contenida en la tabla anterior.

La programación dinámica es un enfoque general para la solución de problemas en los que es necesario tomar decisiones en etapas sucesivas. Las decisiones tomadas en una etapa condicionan la evolución futura del sistema, afectando a las situaciones en las que el sistema se encontrará en el futuro (denominadas estados), y a las decisiones que se plantearán en el futuro. Conviene resaltar que a diferencia de la programación lineal, el modelado de problemas de programación dinámica no sigue una forma estándar. Así, para cada problema será necesario especificar cada uno de los componentes que caracterizan un problema de programación dinámica. El procedimiento general de resolución de estas situaciones se divide en el análisis recursivo de cada una de las etapas del problema, en orden inverso, es decir comenzando por la última y pasando en cada iteración a la etapa antecesora. El análisis de la primera etapa finaliza con la obtención del óptimo del problema. Un ejemplo clásico de un problema que puede resolverse eficientemente mediante la programación dinámica es el cálculo del enésimo término de la sucesión de Fibonacci.

3.3.5 Algoritmos genéticos

Los algoritmos genéticos se presentan con mayor detalle en la sección 2.4. Los algoritmos genéticos son algoritmos matemáticos de optimización de propósito general basados en mecanismos naturales de selección y genética, proporcionando excelentes soluciones a problemas complejos con gran número de parámetros. Son considerados una herramienta muy poderosa de optimización que puede ser usada para resolver una gran número de problemas difíciles con gran eficiencia y exactitud, basándose en la genética natural y la teoría de la evolución de Darwin. Los algoritmos genéticos usan operadores probabilísticos, en vez de los típicos operadores determinísticos de las otras técnicas.

En la naturaleza los individuos de una población compiten entre sí en la búsqueda de recursos tales como comida, agua y refugio. Incluso, los miembros de una misma especie compiten a menudo en la búsqueda de un compañero. Aquellos individuos que tienen más éxito en sobrevivir y en atraer compañeros tienen mayor probabilidad de generar un gran número de descendientes. Por el contrario, individuos poco dotados producirán un menor número de descendientes. Esto significa que los genes de los individuos mejor adaptados se propagarán en sucesivas generaciones hacia un número de individuos creciente. La combinación de buenas características provenientes de diferentes ancestros, puede a veces producir descendientes superiores, cuya adaptación es mucho mayor que la de cualquiera de sus ancestros. De esta manera, las especies evolucionan logrando unas características cada vez mejor adaptadas al entorno en el que viven.

Los algoritmos genéticos usan una analogía directa con el comportamiento natural. Trabajan con una población de individuos, cada uno de los cuales representa una solución factible a un problema dado. A cada individuo se le asigna un valor ó puntuación, relacionado con la calidad de dicha solución. En la naturaleza esto equivaldría al grado de efectividad de un organismo para competir por unos determinados recursos. Cuanto mayor sea la adaptación de un individuo al problema, mayor será la probabilidad de que el mismo sea seleccionado para reproducirse, cruzando su material genético con otro individuo seleccionado de igual forma. Este cruce producirá nuevos individuos, descendientes de los anteriores, los cuales comparten algunas de las características de sus padres. Cuanto menor sea la adaptación de un individuo, menor será la probabilidad de que dicho individuo sea seleccionado para la reproducción, y por tanto de que su material genético se propague en sucesivas generaciones. De esta manera se produce una nueva población de posibles soluciones, la cual reemplaza a la anterior y verifica la interesante propiedad de que contiene una mayor proporción de buenas características en comparación con la población anterior. Así a lo largo de las generaciones las buenas características se propagan a través de la población. Favoreciendo el cruce de los individuos mejor adaptados, van siendo exploradas las áreas más prometedoras del espacio de búsqueda. Si el algoritmo genético ha sido bien diseñado, la población converge hacia una solución óptima del problema.

Los algoritmos genéticos no necesitan conocimientos específicos sobre el problema que intentan resolver y operan de forma simultánea con varias soluciones, en vez de trabajar de forma secuencial como las técnicas tradicionales, por lo que resulta sumamente fácil ejecutarlos en las modernas arquitecturas paralelas. Cuando se usan para problemas de optimización, resultan normalmente menos afectados por los máximos locales que las técnicas tradicionales. Sin embargo, se debe tener mucho cuidado al hacer uso de ellos, puesto que pueden tardar mucho en converger, o no converger en absoluto, dependiendo en cierta medida de los parámetros y operadores que se utili-

cen, como el tamaño de la población, número de generaciones, operadores de cruce, etc.

3.3.6 Algoritmos de búsqueda

La mayoría de problemas de optimización significativos son de complejidad NP-Duros. Este tipo de problemas no es posible enfrentarlos con técnicas determinísticas de enumeración, es decir enumerando todas las posibles soluciones para encontrar la mejor, sino que debemos utilizar técnicas de búsqueda y aproximación, como las que se muestran a continuación.

Búsqueda aleatoria

La búsqueda aleatoria es el método más básico de búsqueda para problemas de optimización. La búsqueda aleatoria pura consiste en generar aleatoriamente un gran número de soluciones a un problema de optimización y seleccionar la mejor de ellas. La búsqueda aleatoria es aplicable prácticamente a cualquier problema de optimización, gracias a la facilidad de implementación que representa y los pocos supuestos que necesita. El esquema básico se observa en el algoritmo 6 para un problema de minimización. Este algoritmo consiste básicamente en generar N soluciones aleatorias que se evalúan según la función objetivo o función de *costo* y si este valor es *mejor* que el óptimo actual, se toma dicha solución como la nueva solución óptima.

Algoritmo 6 Búsqueda aleatoria

1: generar S_0
2: $C_0 = costo(S_0)$
3: $S_{opt} = S_0$
4: $C_{opt} = C_0$
5: $k = 1$
6: **mientras** $k \leq N$ **hacer**
7: generar S_k
8: $C_k = costo(S_k)$
9: **si** $C_k < C_{opt}$ **entonces**
10: $S_{opt} = S_k$
11: $C_{opt} = C_k$
12: **fin si**
13: $k = k + 1$
14: **fin mientras**
15: **devolver** S_{opt}

El método de generación de soluciones aleatorias puede variar con cada problema,

aunque por lo general se usan números aleatorios con distribución uniforme para garantizar que no se da preferencia a ningún tipo de solución.

Búsqueda local

La búsqueda local es uno de los métodos de optimización más sencillos y antiguos. A pesar de su simpleza, ha sido útil con una gran variedad de aplicaciones. El algoritmo como lo plantean Sait y Youssef [1999] inicia con una solución factible inicial $S_0 \in \Omega$ y utiliza una subrutina *mejorar* para buscar una mejor solución en el vecindario de S_0. Si se encuentra una mejor solución $S_1 \in \aleph(S_0)$, entonces la búsqueda continúa en el vecindario $\aleph(S_1)$ de la nueva solución. El algoritmo termina cuando encuentra el óptimo local S_N. El esquema básico se observa en el algoritmo 7. Este algoritmo parte de una solución inicial S_0, la cual, mediante la rutina *mejorar*, busca en el vecindario una *mejor* solución, sobre la cual se analiza un nuevo vecindario para encontrar una nueva mejor solución, y así sucesivamente hasta que no se encuentre una solución mejor en el vecindario de la solución actual. La rutina *mejorar* se comporta de acuerdo con la siguiente definición.

$$mejorar(S) = \begin{cases} T \in \aleph(S) & \text{si el } costo(T) < costo(S) \\ nulo & \text{si no sucede así} \end{cases}$$

Algoritmo 7 Búsqueda local

Entrada: S_0
Salida: S_{opt}
1: $S_2 = S_0$
2: **repetir**
3: $S_1 = S_2$
4: $S_2 = mejorar(S_1)$
5: **hasta que** $S_2 = nulo$
6: **devolver** S_1

La calidad de la solución obtenida por este método depende principalmente de tres aspectos claves:

1. La solución inicial utilizada.
2. La elección o definición del vecindario de una solución.
3. El método de búsqueda sobre el vecindario (i.e. la función *mejorar*).

Por ejemplo, como solución inicial se puede utilizar una buena solución obtenida por algún otro método, una solución completamente aleatoria o incluso correr varias simulaciones con diferentes soluciones iniciales y escoger la mejor solución global que

se encuentre. La elección del vecindario requiere de una buena elección de la función de perturbación. De aquí se deriva el tamaño del vecindario y por ende la velocidad de ejecución del algoritmo y la calidad de la solución. La decisión normalmente se toma tras analizar diferentes experimentos. La subrutina *mejorar* puede explorar todo un vecindario para escoger la mejor solución o simplemente retornar la primera solución que mejore la anterior. Nuevamente, la decisión suele hacerse de manera empírica. El principal problema de la búsqueda local radica en la dificultad de poder elegir correctamente los 3 elementos mencionados. De manera especial, la selección de la solución inicial puede llevarnos a resultados muy diferentes para problemas altamente complejos.

Recocido simulado

El algoritmo de recocido simulado (temple simulado, o en inglés, *simulated annealing* -SA) es uno de los métodos iterativos más utilizados en la solución de problemas de optimización. La principal característica del método es que tiene una probabilidad de aceptar soluciones menos buenas que la anterior, lo que le permite salir de mínimos locales para encontrar mejores soluciones. Esta probabilidad varía en cada iteración, de acuerdo con un concepto denominado la temperatura, que se deriva de la analogía con el proceso físico del temple del acero. Es un algoritmo de búsqueda meta-heurística para problemas de optimización global, es decir, encontrar una buena aproximación al óptimo global de una función en un espacio de búsqueda grande.

El nombre e inspiración viene del proceso de templado del acero, una técnica que consiste en calentar y luego enfriar de forma controlada un material para aumentar el tamaño de sus cristales y reducir sus defectos. El calor causa que los átomos se salgan de sus posiciones iniciales (un mínimo local de energía) y se muevan aleatoriamente; el enfriamiento lento les da mayores probabilidades de encontrar configuraciones con menor energía que la inicial.

En cada iteración, el algoritmo considera algunos vecinos del estado actual S, y probabilísticamente decide entre cambiar el sistema al estado S' o quedarse en el estado S. Las probabilidades se escogen para que el sistema tienda finalmente a estados de menor energía. Típicamente este paso se repite hasta que se alcanza un estado suficientemente bueno para la aplicación o hasta que se cumpla cierto tiempo computacional dado. La probabilidad de hacer la transición al nuevo estado S' es una función $P(\Delta E, T)$ de la diferencia de energía $\Delta E = E(S') - E(S)$ entre los dos estados, y de la variable T, llamada temperatura. Una cualidad importante del método es que la probabilidad de transición P es siempre distinta de cero, aún cuando ΔE sea positivo, es decir, el sistema puede pasar a un estado de mayor energía (peor solución) que el estado actual. Esta cualidad impide que el sistema se quede atrapado en un óptimo local. Cuando la temperatura tiende al mínimo, la probabilidad tiende

a cero asintóticamente. Así, cada vez el algoritmo acepta menos movimientos que aumenten la energía. Si ΔE es negativo, es decir, la transición disminuye la energía, el movimiento es aceptado con probabilidad $P = 1$. Otra característica del algoritmo es que la temperatura va disminuyendo gradualmente conforme avanza la simulación. Hay muchas maneras de disminuir la temperatura, siendo la más usual la exponencial, dónde T disminuye por un factor $\alpha < 1$ en cada paso.

Un algoritmo general de recocido simulado puede plantearse como se muestra en el algoritmo 8. Los datos iniciales y parámetros a ser definidos para poder inicializar el algoritmo, son:

Temperatura inicial (T_0) La temperatura inicial T_0 debe ser una temperatura que permita casi todo movimiento, es decir que la probabilidad de pasar del estado i al j (en $\aleph(i)$) sea muy alta, sin importar la diferencia $c(j) - c(i)$. Esto es que el sistema tenga un alto grado de libertad. En problemas como TSP, donde el input son los nodos de un grafo y las soluciones posibles son distintas formas de recorrer estos nodos, pude tomarse T_0 proporcional a la raíz cuadrada de la cantidad de nodos. En general se toma un valor T_0 que se cree suficientemente alto y se observa la primera etapa para verificar que el sistema tenga un grado de libertad y en función de esta observación se ajusta T_0.

Solución inicial (i_0) La solución factible inicial que llamamos i_0 debería ser una solución tomada al azar del conjunto de soluciones factibles. En algunos problemas esto puede hacerse utilizando números aleatorios provistos por una maquina, pero en muchos casos ya es problemático encontrar una solución, por lo que es imposible tomar una al azar. En estos casos se implementa un algoritmo greedy tipo búsqueda local para buscar una solución factible y se toma esta como i_0.

Función entorno ($\aleph(i)$) Al igual que en el algoritmo de búsqueda local, la selección de un vecindario de una solución es muy importante para que el algoritmo funcione correctamente. Por lo general, el vecindario depende de cada problema, y se genera mediante pequeños cambios en la solución.

Factor de enfriamiento Por lo general se toma un factor de enfriamiento geométrico del tipo $T_1 = \alpha T_0$, con $\alpha < 1$, muy cercano a 1.

Criterio de cambio de la temperatura Se usan dos parámetros: K = cantidad de iteraciones que estamos dispuestos a hacer en cada etapa (equivalente a la cantidad de tiempo que vamos a esperar a que el sistema alcance su equilibrio térmico para una temperatura T); A = cantidad de aceptaciones que se permiten hacer en cada etapa. A medida que T disminuye se supone que al sistema le

resulta más difícil alcanzar un equilibrio porque es más dificultoso el movimiento, entonces hay que esperar más tiempo, esto se traduce en aumentar K.

Criterio de parada En general se utiliza un parámetro de congelamiento. Como a medida que disminuye la temperatura, aumenta el parámetro K y A permanece constante, la proporción A/K se hace pequeña. Asumimos que si $A/K < \epsilon$ el sistema está congelado (la cantidad de aceptaciones respecto de la cantidad de iteraciones es muy chica).

Algoritmo 8 Recocido Simulado

Entrada: S_0, T_0, K_0

1: $S_{opt} = S_0$
2: $T = T_0$
3: $K = K_0$
4: **mientras** $A/K > \epsilon$ **hacer**
5: **mientras** $k < K$ & $a < A$ **hacer**
6: generar S_{ak} en $\aleph(S_{opt})$
7: **si** $C(S_{ak}) - C(S_{opt}) < 0$ **entonces**
8: $S_{opt} = S_{ak}$
9: $a = a + 1$
10: **si no**
11: $r = aleatorio()$
12: **si** $r < e^{(C(S_{opt})-C(S_{ak}))/T}$ **entonces**
13: $S_{opt} = S_{ak}$
14: $a = a + 1$
15: **fin si**
16: **fin si**
17: $k = k + 1$
18: **fin mientras**
19: $T = \alpha T$
20: $K = \rho K$
21: $a = 0$
22: $k = 0$
23: **fin mientras**
24: **devolver** S_{opt}

Búsqueda tabú

Entre los distintos métodos heurísticos de resolución de problemas combinatorios surge, en un intento de dotar de inteligencia a los algoritmos de búsqueda local, el

algoritmo de búsqueda tabú (o TS, de sus siglas del inglés *Tabu Search*), desarrollada por Fred Glover en 1986. La búsqueda tabú, a diferencia de otros algoritmos basados en técnicas aleatorias de búsqueda de soluciones cercanas, se caracteriza porque utiliza una estrategia basada en el uso de estructuras de memoria para escapar de los óptimos locales, en los que se puede caer al moverse de una solución a otra por el espacio de soluciones. Este algoritmo se dota, por tanto, de una memoria donde se almacenan los últimos movimientos realizados, y que puede ser utilizada para recordar aquellos movimientos que hacen caer de nuevo en soluciones ya exploradas. La búsqueda tabú es un método matemático de optimización que hace parte de la clase de técnicas de búsqueda local. La búsqueda tabú mejora el desempeño del método de búsqueda local al utilizar estructuras de memoria. Una vez se determina una solución potencial, ésta se marca como *tabú*, de manera que el algoritmo no visite dicha posibilidad repetidamente.

La búsqueda tabú es un meta-heurístico que puede ser utilizado para resolver problemas de optimización combinatoria como el TSP. El algoritmo utiliza una búsqueda local o de vecindario para moverse iterativamente de una solución S a una solución S' en el vecindario de S ($\aleph(S)$), hasta satisfacer algún criterio de parada. Para evitar caer en mínimos locales, la búsqueda tabú utiliza estructuras de memoria con las que se determinan las soluciones admitidas en $\aleph(S)$. La lista tabú, en su forma más simple, es una memoria de corto plazo que contiene las soluciones que han sido visitadas recientemente (en las últimas n iteraciones). El algoritmo excluye estas soluciones del vecindario $\aleph(S)$. Un algoritmo básico de búsqueda tabú [Zolfaghari y Liang, 2002] se presenta en el algoritmo 9.

Además del uso de una lista tabú, la búsqueda tabú introduce los conceptos de intensificación y diversificación en los algoritmos de búsqueda. La idea detrás del concepto de *intensificación* es que se debe explorar más a fondo en partes del espacio de búsqueda que parecen más prometedoras para asegurarse de encontrar las mejores soluciones de dichas áreas. Así, se pasa algún tiempo intensificando la búsqueda en una misma región, antes de pasar a un espacio de solución más lejano. Por otro lado, el concepto de *diversificación* pretende evitar que el algoritmo caiga en óptimos locales, forzando al algoritmo a buscar en áreas inexploradas del espacio de búsqueda, basado en una memoria de largo plazo. La correcta definición de estos dos elementos es un punto crítico en el desempeño del algoritmo.

Algoritmo 9 Búsqueda Tabú

1: *Inicialización.* Leer el tamaño de la lista tabú (l_T) y el intervalo de diversificación (i_D). Iteración $t = 0$.
2: *Solución inicial.* Generar una solución inicial y calcular el valor de la función objetivo. Hacer la solución actual y el óptimo global iguales a la solución inicial.
3: *Generación de vecindario.* Generar todas las posibles soluciones del vecindario mediante pequeños cambios en la actual. Calcular los valores de la función objetivo.
4: *Búsqueda en el vecindario.* Escoger la mejor solución no-tabú en el vecindario como solución candidata. Si el valor de la solución candidata es mejor que el óptimo global, hacer óptimo global igual a la solución candidata.
5: *Aspiración.* Si no se puede seleccionar una nueva solución, ignorar la lista tabú y seleccionar como candidata a la mejor solución del vecindario.
6: *Actualizar lista tabú.* Agregar la solución actual a la lista tabú. Si la lista tiene l_T elementos, eliminar el más antiguo.
7: *Mover.* Hacer la solución actual igual a la solución candidata e incrementar el contador de iteraciones.
8: *Diversificación.* Si $t \bmod i_D = 0$, se cumplió el intervalo de diversificación, entonces ir a paso 2, si no, ir al paso 9.
9: *Terminación.* Si se cumplen el criterio de parada, terminar, si no, ir al paso 3.

Capítulo 4

Caso de estudio

Como ya se ha dicho anteriormente, el problema estudiado durante este proyecto es el problema de corte en una dimensión, aplicado específicamente a la industria de cartón, para la automatización y optimización del proceso de corte de cartón para elaborar cajas. En general, el problema de corte [Yen *et al.*, 2004] (roll-trim o cutting-stock) es un problema de programación entera donde el objetivo principal es recortar productos de algún material (e.g. papel, cartón, tela) de diferentes tamaños a partir de un rollo o una lámina larga, con el fin de cumplir las órdenes o pedidos realizados por los clientes. Los pedidos se hacen por diferentes cantidades y con distintas dimensiones del producto, por lo cual un grupo de pedidos generalmente se ejecuta con desperdicio de una parte del material. El esquema óptimo de corte es el que minimiza el desperdicio de material. El planteamiento del problema se hace bajo las condiciones y restricciones propias del modelo específico de una empresa.

4.1 Planteamiento del problema

Se cuenta con láminas de cartón de un ancho específico y una longitud que para efectos del planteamiento teórico se considera infinita. A partir de estas láminas se deben recortar rectángulos para armar cajas de diferentes tamaños, según los pedidos realizados por los clientes. Los n pedidos de cajas se especifican con el ancho W_i (*width*) y largo H_i (*height*) del rectángulo, así como el número de cajas requeridas Q_i (*quantity*) y un código de identificación para distinguir cada pedido $i = 1, 2, \ldots, n$.

Los pedidos se introducen al sistema para generar los patrones $j = 1, 2, \ldots, m$ que se utilizan para cortar el material. Un *patrón de corte* consiste en fijar en una cierta posición las cuchillas de la cizalla (máquina de corte) para obtener tiras de cartón con anchos específicos, y se define mediante los códigos de identificación de los dos pedidos que hacen parte del patrón y el número de tiras de cada pedido que lo

conforman. Con el patrón seleccionado se recorta la lámina en tiras de anchos W_i y $W_{i'}$ de los pedidos i e i'. Los cortes transversales sobre las tiras resultantes en este proceso son realizados por una máquina diferente y no hacen parte del problema de optimización. La primera máquina, cuya programación se desea optimizar, cuenta con 7 cuchillas que generan un máximo de 6 tiras de cartón por corte y eliminan los sobrantes o desperdicio T (*trim-loss*). Por las restricciones físicas de la máquina, las 6 tiras recortadas en cualquier momento solo pueden ser de 2 anchos diferentes. El proceso de corte se ilustra en la figura 4.1.

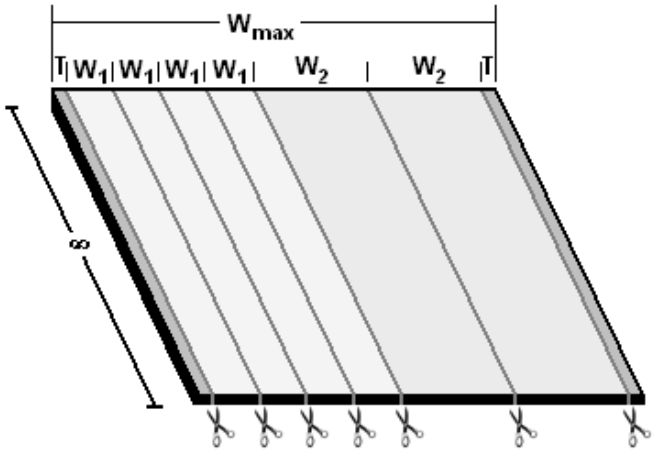


Figura 4.1: Problema de corte unidimensional.

Para realizar el cambio de un patrón de corte a otro diferente se requiere detener la cizalla y cambiar la posición de las cuchillas, proceso que toma un tiempo considerable y en el cual se suspende el proceso de corte. Por otro lado, en este sector industrial las cantidades ordenadas por los clientes permiten un margen de error del 10%; es decir, se puede entregar al cliente cualquier cantidad de cajas entre el 90% y 110% de la cantidad ordenada.

El problema consiste entonces en minimizar el desperdicio de cartón y los cambios de posición de las cuchillas, organizando de la mejor manera posible los pares de cajas que se programarán en la máquina. Denominamos *programación* al proceso que se realiza para fijar la posición de las cuchillas en la máquina. Para ello vamos a encontrar el valor x_j correspondiente a los centímetros de lámina que se procesan con cada patrón j para minimizar el costo asociado al desperdicio de material y al tiempo muerto (ecuación 4.1). A la cantidad total que vamos a hallar le asignamos la incógnita $\mathbf{X} = (x_1, x_2, \ldots, x_m)$ correspondiente a los m patrones procesados en una programación. La cantidad procesada de cada pedido debe estar entre un 90% y

110% del pedido (ecuación 4.2), y cada patron definido debe ajustarse al ancho W_{max} de la lámina (ecuación 4.3). El modelo matemático que planteamos es el siguiente:

$$\min C_T \sum_{j=1}^{m} T_j x_j + C_D \sum_{j=1}^{m} \delta(x_j) \tag{4.1}$$

Sujeto a las restricciones:

$$0.9HQ_i \leq \sum_{j=1}^{m} a_{ij} x_j \leq 1.1HQ_i \text{ (para } i = 1, 2, \ldots, n) \tag{4.2}$$

$$\sum_{i=1}^{n} a_{ij} W_i \leq W_{max} \text{ (para } j = 1, 2, \ldots, m) \tag{4.3}$$

$$a_{ij} \geq 0 \tag{4.4}$$

$$x_j \geq 0 \tag{4.5}$$

Donde:

i Representa un pedido. $i = 1, 2, \ldots, n$.

j Representa un patrón. $j = 1, 2, \ldots, m$.

a_{ij} Elementos de pedido i en el patrón j.

x_j Centímetros de lámina procesados con el patrón j.

T_j Centímetros de desperdicio en el patrón j.

W_i Ancho del pedido i.

W_{max} Ancho máximo de las láminas de cartón.

HQ_i Total de centímetros necesarios para armar Q cajas del pedido i con largo H.

C_T Costo por centímetro cuadrado de desperdicio.

C_D Costo por cambio de posición en las cuchillas para procesar un nuevo patrón.

$$\delta(x_j) = \begin{cases} 1 & \text{si se usa el patrón } j \\ 0 & \text{si no se usa} \end{cases}$$

4.2 Solución mediante programación lineal

Como un primer acercamiento a la solución del problema se ha utilizado la programación lineal con diferentes modelos del problema, que varían según la formulación de la función objetivo y las restricciones.

4.2.1 Primera formulación - Método PL1

Como primera formulación del problema en forma de programación lineal, definimos el siguiente modelo, con el que se pretende minimizar el desperdicio total (ecuación 4.6), restringiendo la programación de todos los pedidos a estar entre un 90% y un 110% del total de la orden.

$$\min \sum_{j=1}^{m} T_j x_j \tag{4.6}$$

sujeto a

$$\begin{aligned}
a_{11}x_1 + a_{12}x_2 + \cdots + a_{1m}x_m &\leq 1.1HQ_1 \\
a_{21}x_1 + a_{22}x_2 + \cdots + a_{2m}x_m &\leq 1.1HQ_2 \\
&\vdots \\
a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nm}x_m &\leq 1.1HQ_n \\
-a_{11}x_1 - a_{12}x_2 - \cdots - a_{1m}x_m &\leq -0.9HQ_1 \\
-a_{21}x_1 - a_{22}x_2 - \cdots - a_{2m}x_m &\leq -0.9HQ_2 \\
&\vdots \\
-a_{n1}x_1 - a_{n2}x_2 - \cdots - a_{nm}x_m &\leq -0.9HQ_n \\
x_1, x_2, \ldots, x_m &\geq 0
\end{aligned}$$

donde además

$$\sum_{i=1}^{n} a_{ij} W_i \leq W_{max} \text{ (para } j = 1, 2, \ldots, m) \tag{4.7}$$

El procedimiento básico utilizado en esta primera formulación se presenta en el algoritmo 10. En primera instancia se cargan los datos de entrada correspondientes al *ancho máximo* (W_{max}) de las láminas de cartón y la información sobre los *pedidos* de los clientes, incluyendo el código de identificación (ID), el ancho (W), el largo (H) y la cantidad de cajas pedidas (Q). A partir de este arreglo se puede saber fácilmente el número total de pedidos, n. Teniendo estos datos se generan la matriz $\mathbf{A}$ y los vectores $\mathbf{T}$ y $\mathbf{b}$ utilizados en el planteamiento del problema de programación lineal. La matriz $\mathbf{A}$ corresponde a los coeficientes de las restricciones, donde cada elemento a_{ij} representa el número de tiras del pedido i que se generan al utilizar el patrón j. El vector $\mathbf{b}$ corresponde al vector del lado derecho de las restricciones. En este caso, cada restricción i se refiere al total de centímetros $P_i = a_{i1}x_1 + a_{i2}x_2 + \cdots + a_{im}x_m$ que deben procesarse del pedido i, el cual como se explicó anteriormente, debe cumplir el total de la orden (expresado en centímetros como HQ_i) con un margen de error del 10%, es decir, $0.9HQ_i \leq P_i \leq 1.1HQ_i$. Como el problema debe llevarse a la forma general de un problema de programación lineal, todas las restricciones deben ser de la

forma $\leq$, por lo cual se debe tener dos restricciones para cada pedido, una de la forma $P_i \leq 1.1HQ_i$, y otra de la forma $-P_i \leq -0.9HQ_i$, equivalente a tener $P_i \geq 0.9HQ_i$. Finalmente, el vector **T** corresponde al vector de coeficientes de la función objetivo, que en nuestro caso representa el desperdicio lineal T_j generado por cada patrón j, y al ser multiplicado por la cantidad de centímetros procesados con dicho patrón, se obtiene el desperdicio total en centímetros cuadrados.

Algoritmo 10 Programación Lineal - primera formulación.

Entrada: $pedidos(ID, W, H, Q)$, W_{max}

1: $n = count(pedidos)$
2: generar T, A y b (* rutina para generar las matrices necesarias *)
3: $X = linprog(T, A, b)$ (* resolver el problema de programación lineal *)
4: **devolver** X

Para llevar a cabo la rutina *generar* que se ejecuta en la segunda línea del algoritmo se han evaluado dos opciones. En la primera opción, que se muestra en el algoritmo 11, se producen todas las posibles combinaciones de pedidos válidas para formar patrones cuyo ancho sea menor o igual al ancho máximo de la lámina. En este caso, el total de patrones generados y que pueden ser utilizados es cercano a $6n^2$. Por ejemplo, en el caso de tener 100 pedidos que deben ser programados, este método genera 59900 patrones posibles, de los cuales en la solución óptima con el menor desperdicio se utilizan 12000.

Como segunda opción, con el objetivo de acotar el problema, se utilizaron únicamente $m = n(n+1)/2$ patrones, correspondientes a la mejor combinación (mínimo desperdicio) que se puede lograr con cada par de pedidos. Es decir, de todos los posibles patrones que se puedan formar con dos pedidos específicos, se toma el que tenga el menor desperdicio. Mediante esta aproximación se reduce bastante el número de patrones generados, y a su vez, el número de patrones utilizados. En este caso, para el mismo ejemplo de 100 pedidos, se generan 5050 patrones, los cuales se utilizan todos en la solución óptima. Además, el porcentaje total de desperdicio es muy similar con los dos métodos.

Independiente del método de generación para **T**, **A** y **b**, con este método se encuentra siempre una combinación que minimiza el desperdicio de cartón, sujeto a cumplir con la totalidad de los pedidos dentro del margen permitido. Sin embargo, los resultados obtenidos no fueron del todo satisfactorios, ya que se está obligando al sistema a programar el total de los pedidos de los clientes en una sola corrida, cuando en realidad los pedidos de los clientes no tienen que ser todos evacuados de inmediato, y podrían en cambio dejarse algunos pedidos pendientes, a la espera que puedan programarse posteriormente cuando nuevas órdenes ingresen al sistema. Debido a esta restricción, el desperdicio en algunos de los patrones utilizados es inaceptablemente

Algoritmo 11 Programación Lineal - generar T, A, b - primer método.

Entrada: $pedidos(ID, W, H, Q)$, W_{max}, n

1: $t = 1$
2: **para** $i = 1$ hasta n **hacer**
3: $b_i = 1.1 \cdot pedidos_i(H) \cdot pedidos_i(Q)$
4: $b_{n+i} = -0.90 \cdot pedidos_i(H) \cdot pedidos_i(Q)$
5: **para** $j = i$ hasta n **hacer**
6: **para** $k = 1$ hasta 6 **hacer**
7: **para** $l = 0$ hasta $6 - k$ **hacer**
8: $W_{comb} = k \cdot pedidos_i(W) + l \cdot pedidos_j(W)$
9: **si** $W_{max} \geq W_{comb}$ **entonces**
10: $T_t = W_{max} - W_{comb}$
11: **si** $i = j$ **entonces**
12: $A_{it} = k + l$
13: **si no**
14: $A_{it} = k$
15: $A_{jt} = l$
16: **fin si**
17: **fin si**
18: **fin para**
19: $t = t + 1$
20: **fin para**
21: **fin para**
22: **fin para**
23: **devolver** T, A, b

Algoritmo 12 Programación Lineal - generar T, A, b - segundo método.

Entrada: $pedidos(ID, W, H, Q)$, W_{max}, n

1: $t = 1$
2: **para** $i = 1$ hasta n **hacer**
3: $b_i = 1.1 \cdot pedidos_i(H) \cdot pedidos_i(Q)$
4: $b_{n+i} = -0.90 \cdot pedidos_i(H) \cdot pedidos_i(Q)$
5: **para** $j = i$ hasta n **hacer**
6: **para** $k = 1$ hasta 6 **hacer**
7: **para** $l = 0$ hasta $6 - k$ **hacer**
8: $W_{comb} = k \cdot pedidos_i(W) + l \cdot pedidos_j(W)$
9: **si** $W_{max} - W_{comb} < T_t$ AND $W_{max} \geq W_{comb}$ **entonces**
10: $T_t = W_{max} - W_{comb}$
11: **si** $i = j$ **entonces**
12: $A_{it} = k + l$
13: **si no**
14: $A_{it} = k$
15: $A_{jt} = l$
16: **fin si**
17: **fin si**
18: **fin para**
19: **fin para**
20: $t = t + 1$
21: **fin para**
22: **fin para**
23: **devolver** T, A, b

elevado.

En otras palabras, con este planteamiento se programan todos los pedidos y se busca el desperdicio mínimo. Sin embargo la solución no es la mejor, ya que hay pedidos que generan un gran desperdicio y se pueden dejar para ser programados después y de esta manera cumplir de momento los pedidos que dejan menor desperdicio. Además, al minimizar el desperdicio, la mayoría de pedidos se programan cumpliendo con el 90% de la cantidad pedida, lo que conlleva a una disminución en las unidades vendidas por la empresa, y por ende en las utilidades.

Para solucionar este problema se podría pensar que basta con restringir los patrones a aquellos que tengan un desperdicio por debajo del límite aceptable. Sin embargo, si el resto del planteamiento permanece igual, no se puede asegurar que exista siempre una respuesta para el problema de programación lineal con las restricciones y coeficientes que quedan, ya que el cumplimiento mínimo de cada pedido puede que no siempre se cumpla, por lo que no es posible solucionarlo de esta manera. Si por otro lado se suprime la restricción del mínimo de cajas que se requieren para cada pedido y se mantienen la restricción del máximo y la función de minimización del desperdicio, entonces la respuesta obtenida sería $\mathbf{X} = \mathbf{0}$, para todos los patrones, ya que el mínimo desperdicio ocurre cuando no se utiliza ningún patrón y no se hace ningún corte.

Ejemplo

Para ilustrar el comportamiento de este método, planteamos el siguiente ejemplo, con una muestra de solamente 5 pedidos, como se muestra en la tabla 4.1. El método de generación de patrones utilizados en este caso será el segundo método (algoritmo 12). Para el ejemplo se asume un ancho máximo de las láminas, $W_{max} = 197cm$.

Tabla 4.1: Ejemplo con 5 pedidos de cajas.

Pedido	Ancho	Largo	Cantidad
1	65,2	114,3	4000
2	74,2	74,2	6000
3	108,9	96,9	2500
4	85,4	125,7	3000
5	48,1	131,9	5000

Al utilizar el segundo método de *generación de patrones*, podemos generar un total de 15 patrones, $m = n(n+1)/2 = 5(6)/2 = 15$. Los patrones generados corresponden a la mejor combinación (menor desperdicio) entre cada par de pedidos y se pueden apreciar en la tabla 4.2. El significado de las columnas de la tabla es el siguiente: *Patrón* es el número de identificación del patrón, $j = 1, 2, \ldots, m$. Las

columnas ID_1 e ID_2 hacen referencia al identificador de los dos pedidos utilizados en cada patrón. Q_1 y Q_2 son la cantidad de elementos de cada uno de estos pedidos que conforman el patrón. La última columna, denotada con T, son los centímetros lineales de desperdicio que se producen al utilizar cada patrón. Por ejemplo, con el patrón 9, se recortan dos tiras con el ancho del pedido 2 (de 74,2cm) y una tira con el ancho del pedido 5 (de 48,1cm), para un ancho total de 196,5cm que genera tan solo 0,5cm de desperdicio al ser utilizado.

Tabla 4.2: Patrones generados para el ejemplo de 5 pedidos.

Patrón	ID_1	Q_1	ID_2	Q_2	T
1	1	3	1	0	1,4
2	1	1	2	1	57,6
3	1	1	3	1	22,9
4	1	1	4	1	46,4
5	1	2	5	1	18,5
6	2	2	2	0	48,6
7	2	1	3	1	13,9
8	2	1	4	1	37,4
9	2	2	5	1	0,5
10	3	1	3	0	88,1
11	3	1	4	1	2,7
12	3	1	5	1	40,0
13	4	2	4	0	26,2
14	4	1	5	2	15,4
15	5	4	5	0	4,6

A partir de las tablas 4.1 y 4.2 se pueden construir la matriz **A** y los vectores **T** y **b**. El planteamiento del problema de programación lineal se presenta a continuación.

Función objetivo:

$$\begin{aligned}\min \quad & 1.4x_1 + 57.6x_2 + 22.9x_3 + 46.4x_4 + 18.5x_5 + 48.6x_6 + 13.9x_7 + 37.4x_8 \\ & +0.5x_9 + 88.1x_{10} + 2.7x_{11} + 40x_{12} + 26.2x_{13} + 15.4x_{14} + 4.6x_{15}\end{aligned}$$

sujeto a

$$\begin{aligned}3x_1 + 1x_2 + 1x_3 + 1x_4 + 2x_5 &\leq 1.1(457200) \\ 1x_2 + 2x_6 + 1x_7 + 1x_8 + 2x_9 &\leq 1.1(445200) \\ 1x_3 + 1x_7 + 1x_{10} + 1x_{11} + 1x_{12} &\leq 1.1(242250) \\ 1x_4 + 1x_8 + 1x_{11} + 2x_{13} + 1x_{14} &\leq 1.1(377100) \\ 1x_5 + 1x_9 + 1x_{12} + 2x_{14} + 4x_{15} &\leq 1.1(659500)\end{aligned}$$

$$
\begin{aligned}
-3x_1 - 1x_2 - 1x_3 - 1x_4 - 2x_5 &\leq -0.9(457200) \\
-1x_2 - 2x_6 - 1x_7 - 1x_8 - 2x_9 &\leq -0.9(445200) \\
-1x_3 - 1x_7 - 1x_{10} - 1x_{11} - 1x_{12} &\leq -0.9(242250) \\
-1x_4 - 1x_8 - 1x_{11} - 2x_{13} - 1x_{14} &\leq -0.9(377100) \\
-1x_5 - 1x_9 - 1x_{12} - 2x_{14} - 4x_{15} &\leq -0.9(659500) \\
x_1, x_2, \ldots, x_{15} &\geq 0
\end{aligned}
$$

Con el problema planteado como programación lineal, se puede utilizar cualquiera de los muchos paquetes y herramientas que existen para resolver este tipo de problemas, todos ellos basados en el algoritmo Símplex, explicado en la sección 2.3. En nuestro caso hemos utilizado las funciones de Matlab. Después de ejecutar el algoritmo, la solución óptima que obtenemos es la que se presenta en la tabla 4.3. En dicha tabla se muestra cada patrón generado, el desperdicio T producido al ser utilizado y el total de centímetros X de cartón que se deben procesar con cada patrón para minimizar el total de desperdicio. Como podemos apreciar, los patrones que menor desperdicio generan son los más utilizados (patrones 9, 1, 11 y 15), además de los patrones 13 y 14, los cuales producen un desperdicio mayor, pero fueron necesarios para completar todas las órdenes. Se puede también observar que de los 15 patrones generados, solamente fueron utilizados 6 en la solución óptima.

Tabla 4.3: Primera solución por PL para el ejemplo con 5 pedidos.

Patrón	T	X
1	1,4	137160
2	57,6	0
3	22,9	0
4	46,4	0
5	18,5	0
6	48,6	0
7	13,9	0
8	37,4	0
9	0,5	244860
10	88,1	0
11	2,7	266470
12	40	0
13	26,2	20190
14	15,4	32530
15	4,6	70910

En la tabla 4.4 se muestra el cumplimiento de las órdenes que se obtiene mediante

el uso de la solución óptima encontrada. Se puede apreciar que todos los pedidos se cumplen dentro del margen del 10% permitido. En este caso el porcentaje total programado fue un 98% de la cantidad ordenada. En total se utilizaron 6 patrones y el desperdicio de material es del 1.6%. Se observa también que los pedidos 1, 4, y 5 se cumplen en el 90% y los pedidos 2 y 3 se cumplen al 110%.

Tabla 4.4: Cumplimiento de las órdenes con la primera solución por PL.

Pedido	Ordenado	Producido
1	4000	3600
2	6000	6600
3	2500	2750
4	3000	2700
5	5000	4500
Total	**20500**	**20150**

4.2.2 Segunda formulación - Método PL2

Buscando mejorar los resultados obtenidos en la primera aproximación, se cambió la formulación del problema y utilizamos nuevamente la programación lineal para su solución. Esta vez suponemos que no es indispensable cumplir con todas las órdenes en una sola corrida, puesto que mientras algunas órdenes son procesadas pueden llegar nuevos pedidos de los clientes que permitan mejores combinaciones que los iniciales. Con esta segunda formulación se pretende maximizar la longitud de los patrones programados en su totalidad (ecuación 4.8), de tal manera que la cantidad de unidades procesadas no supere el 110% de las pedidas, cumpliendo así la mayor cantidad posible de los pedidos. El planteamiento utilizado es el siguiente:

$$\max \sum_{j=1}^{m} x_j \tag{4.8}$$

sujeto a

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1m}x_m &\leq 1.1HQ_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2m}x_m &\leq 1.1HQ_2 \\ &\vdots \\ a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nm}x_m &\leq 1.1HQ_n \\ x_1, x_2, \ldots, x_n &\geq 0 \end{aligned}$$

En este caso se restringen los patrones utilizados a aquellos que produzcan un desperdicio menor a un máximo permitido, asegurando así la minimización del desperdicio. Al comparar este método con el anterior obtenemos una reducción del

desperdicio bastante significativa, gracias al preprocesamiento que se hace para seleccionar únicamente los patrones que produzcan un desperdicio inferior a un máximo permitido. El esquema general en este caso se comporta igual a la primera aproximación, planteada en el algoritmo 10. La rutina para generar la matriz **A** y el vector **b** se muestra en el algoritmo 13.

Algoritmo 13 Programación Lineal - segunda formulación (generar pedidos).

Entrada: $pedidos(ID, W, H, Q)$, W_{max}, W_{min}, n

```
1:  t = 1
2:  para i = 1 hasta n hacer
3:    b_i = 1.1 · pedidos_i(H) · pedidos_i(Q)
4:    para j = i hasta n hacer
5:      para k = 1 hasta 5 hacer
6:        para l = 1 hasta 6 − k hacer
7:          W_comb = k · pedidos_i(W) + l · pedidos_j(W)
8:          si W_min ≤ W_comb AND W_max ≥ W_comb entonces
9:            T_t = W_max − W_comb
10:           si i = j entonces
11:             A_it = k + l
12:           si no
13:             A_it = k
14:             A_jt = l
15:           fin si
16:         fin si
17:         t = t + 1
18:       fin para
19:     fin para
20:   fin para
21: fin para
22: devolver A, b
```

Ejemplo

Para ilustrar el comportamiento de este método hemos utilizado el mismo ejemplo anterior, cuyos datos se muestran en la tabla 4.1. El ancho mínimo se ha establecido en 192cm y el ancho máximo se ha dejado en el mismo valor de 197cm, con lo cual se establece un desperdicio máximo de 5cm lineales, equivalente a un 2.5%. En la tabla 4.5 se muestran los patrones que cumplen esta restricción, correspondientes a los patrones 1, 9, 11 y 15 del ejemplo anterior.

Tabla 4.5: Patrones generados para el ejemplo de 5 pedidos.

Patrón	ID_1	Q_1	ID_2	Q_2	T
1	1	3	1	0	1,4
2	2	2	5	1	0,5
3	3	1	4	1	2,7
4	5	4	5	0	4,6

A partir de las tablas 4.1 y 4.5 se pueden construir la matriz **A** y el vector **b**. En este caso no se utiliza el vector **T** por la forma como se define la función objetivo, donde todos los coeficientes son 1. Sin embargo, como el problema se presenta en forma de maximización, debemos transformarlo en un problema general de programación lineal. Para ello, la función objetivo $\max f$ se sustituye por $\min -f$, y y se resuelve como un problema de minimización. El planteamiento del problema de programación lineal se presenta a continuación.

Función objetivo:

$$\min -x_1 - x_2 - x_3 - x_4 \tag{4.9}$$

sujeto a

$$\begin{aligned}
3x_1 + 0x_2 + 0x_3 + 0x_4 &\leq 1.1(457200) \\
0x_1 + 2x_2 + 0x_3 + 0x_4 &\leq 1.1(445200) \\
0x_1 + 0x_2 + 1x_3 + 0x_4 &\leq 1.1(242250) \\
0x_1 + 0x_2 + 1x_3 + 0x_4 &\leq 1.1(377100) \\
0x_1 + 1x_2 + 0x_3 + 4x_4 &\leq 1.1(659500) \\
x_1, x_2, \ldots, x_{15} &\geq 0
\end{aligned}$$

Con el problema planteado como programación lineal, podemos nuevamente utilizar una herramienta para resolverlo automáticamente. Después de ejecutar el algoritmo, la solución óptima que obtenemos es la mostrada en la tabla 4.6. Como podemos apreciar, los 4 patrones generados fueron utilizados.

Tabla 4.6: Segunda solución por PL para el ejemplo con 5 pedidos.

Patrón	T	X
1	1,4	167640
2	0,5	244860
3	2,7	266470
4	4,6	120150

En la tabla 4.7 se muestra el cumplimiento de las órdenes que se obtiene mediante

el uso de la solución óptima encontrada. Se puede apreciar que en este caso no todos los pedidos se cumplen dentro del margen del 10% permitido, ya que del pedido número 4 se produce sólo un 70% de las unidades ordenadas, dejando 880 unidades pendientes para una posterior programación. En este caso el porcentaje total programado fue un 104% de la cantidad ordenada. En total se utilizaron 4 patrones y el desperdicio de material es del 1.05%. Además, los cuatro pedidos que se finalizan se están cumpliendo al 110%.

Tabla 4.7: Cumplimiento de las órdenes con la segunda solución por PL.

Pedido	Ordenado	Producido	Pendiente
1	4000	4400	0
2	6000	6600	0
3	2500	2750	0
4	3000	2120	880
5	5000	5500	0
Total	**20500**	**21370**	**880**

Con este ejemplo, el segundo método presenta algunas ventajas frente al primero. En primer lugar, el número de patrones utilizados es menor, haciendo uso de cuatro patrones frente a seis que se utilizaron con el primer acercamiento. Menos patrones utilizados significan menos cambios en la configuración de la cizalla, y por tanto, mayor tiempo de producción sin detener la máquina. En segundo lugar, la cantidad total de unidades producidas es mayor en el segundo caso. Aunque es de esperar que no siempre el total producido sea mayor con este método que con el primero, lo que sí podemos esperar es que cada pedido se intenta programar al máximo permitido, es decir al 110% para maximizar la producción, mientras que en el primer caso se programa cercano al mínimo permitido, el 90%, con el fin de minimizar el desperdicio total. Finalmente, la ventaja más importante que arroja este ejemplo para el segundo método es que el desperdicio total se ha reducido del 1.60% al 1.05%, lo cual en este caso equivale aproximadamente a $76m^2$ de material.

4.3 Solución mediante algoritmos genéticos

Los algoritmos genéticos [Coley, 1999] son métodos de optimización numérica inspirados en la genética, diseñados en los años setenta por John Holland. Su funcionamiento se basa en la evolución y la biología. Estos algoritmos obtienen la solución mediante la evolución de una población de individuos sometida a acciones aleatorias semejantes a las que actúan en la evolución biológica (selección natural, combinación genética y mutaciones). Aunque la solución encontrada mediante programación lineal parece

buena, no se tienen en cuenta todas las variables que se deben optimizar: minimizar los cambios de cuchillas, minimizar el desperdicio y maximizar el cumplimiento de pedidos sin incurrir en excesos de producción sobre el margen de 10% permitido. Con este fin, hemos decidido hacer uso de los Algoritmos Genéticos, para buscar una solución más cercana a las condiciones reales del problema, donde se tengan en cuenta todas las variables mencionadas.

4.3.1 Primera formulación - Método AG1

La *representación* que hemos definido para los individuos tiene la misma estructura que la utilizada en la programación lineal, es decir un vector $\mathbf{X} = (x_1, x_2, \ldots, x_m)$, donde cada x_j representa la cantidad en centímetros de lámina que se van a recortar siguiendo el patrón j. Los patrones están restringidos al igual que en la segunda aproximación de la programación lineal, es decir solo se utilizan aquellos que cumplan con un desperdicio menor a una cantidad dada.

La *población inicial* se genera de manera aleatoria. Primero, calculamos el valor $C_{mut} = \overline{HQ} \cdot n/m$ que se utiliza también en la función de mutación. Este valor corresponde a la media aritmética del vector HQ, multiplicada por el factor entre el número de pedidos n y el número de patrones posibles m. Luego se crea la población inicial con valores aleatorios entre 0 y 1, multiplicados por el valor calculado de C_{mut}.

Como se ha dicho anteriormente, la *función objetivo* debe incluir las variables que midan el desperdicio, el cumplimiento de los pedidos y los cambios de cuchillas. Además, como no existe una restricción para la cantidad máxima procesada con un patrón, también se debe castigar el desperdicio por exceso de corte. En una primera aproximación se utiliza la función objetivo que muestra la ecuación 4.10, también llamada función de *adaptación*, o *fitness value*.

Sean

$\sum_{i=1}^{n} |HQ_i - \mathbf{A}_{i*} \times \mathbf{X}|$ La distancia de cubrimiento de los pedidos. Es decir, qué tan lejos de cumplir el 100% de los pedidos está cada individuo de la población.

$\sum_{j=1}^{m} T_j x_j$ El desperdicio total en centímetros cuadrados.

$\text{nnz}(\mathbf{X})$ El número de cambios de cuchilla o patrones utilizados (nnz es el número de elementos diferentes de cero en una matriz o vector).

Además, a cada término le damos diferentes pesos dentro de la función, multiplicando por constantes que hagan más importante uno u otro ítem de optimización. La función objetivo resultante tiene la forma

$$\alpha \sum_{i=1}^{n} |HQ_i - \mathbf{A}_{i*} \times \mathbf{X}| + \beta \sum_{j=1}^{m} T_j x_j + \gamma \cdot k_t \cdot \text{nnz}(\mathbf{X}) \tag{4.10}$$

El último término se multiplica además por una constante k_t para convertir el número de patrones utilizados en los centímetros de material que se están dejando de procesar en el tiempo muerto de la máquina. Para diferentes valores de α, β, γ los resultados cambian enormemente. Con un valor muy alto en el primer término, el desperdicio se puede hacer muy grande, mientras que si se le da un mayor peso al último término, la cantidad de material procesada será mucho menor. En las pruebas se hicieron experimentos con diferentes valores, y aunque los resultados no fueron del todo malos, se pensó en una nueva función objetivo, donde se visualiza mejor el cumplimiento de los diferentes objetivos de optimización. El esquema general del algoritmo genético utilizado se muestra en el algoritmo 14.

Algoritmo 14 Algoritmo Genético.

Entrada: $pedidos(ID, W, H, Q)$, W_{max}, W_{min}
Entrada: N_{gen}, N_{ind}, P_m, P_{elite}
1: $n = count(pedidos)$
2: $N_{elite} = ceil(N_{ind}P_{elite}/100)$
3: generar T, A y HQ (* rutina para generar las matrices necesarias *)
4: $cromtam = count(T)$ (* tamaño del cromosoma *)
5: $maxcosto = max(T)$ (* desperdicio máximo de un patrón *)
6: $TotPedidos = HQ \cdot W$ (* total de cm^2 de los pedidos *)
7: $C_{mut} = mean(pedidos_i(H) \cdot pedidos_i(Q) \cdot n/m$
8: $Pop_{ini} = rand(N_{ind}, cromtam) \cdot Cmut$ (* población inicial *)
9: $Pop_{old} = Pop_{ini}$
10: call $fitness()$
11: $gen = 0$ (* iniciar contador de generaciones *)
12: **mientras** $gen < N_{gen}$ **hacer**
13: $Pop_{new} = Pop_{old}[1 : N_{elite}]$
14: call $cruce()$ (* función de cruce *)
15: call $mutacion()$ (* función de mutación *)
16: call $fitness()$ (* función de fitness *)
17: $sort(Pop_{old})$ (* ordenar la población por el fitness *)
18: $gen = gen + 1$ (* incrementar contador de generaciones *)
19: **fin mientras**
20: **devolver** $Pop_{old}[1]$

El algoritmo recibe como entrada los datos básicos de las órdenes o pedidos, además del ancho mínimo y máximo que pueden tener los patrones. También recibe los parámetros básicos del algoritmo genético, como el número máximo de generaciones para el criterio de parada (N_{gen}), el número de individuos en la población (N_{ind}), la probabilidad de mutación P_m y el porcentaje de elitismo, P_{elite}. A partir de estos

datos se generan la matriz **A** y los vectores **T** y **HQ**. Además, se calcula el tamaño del cromosoma (número de patrones generados) y el valor C_{mut} que se utiliza en la función de mutación y al generar la población inicial. La generación de **A**, **T** y **HQ** se presenta en el algoritmo 15, que es similar a la de la programación lineal, algoritmo 13.

Algoritmo 15 Algoritmo Genético - generar matrices.

Entrada: $pedidos(ID, W, H, Q)$, W_{max}, W_{min}, n

```
1: t = 1
2: para i = 1 hasta n hacer
3:    HQ_i = pedidos_i(H) · pedidos_i(Q)
4:    para j = i hasta n hacer
5:       para k = 1 hasta 5 hacer
6:          para l = 1 hasta 6 − k hacer
7:             W_comb = k · pedidos_i(W) + l · pedidos_j(W)
8:             si W_min ≤ W_comb AND W_max ≥ W_comb entonces
9:                T_t = W_max − W_comb
10:               si i = j entonces
11:                  A_it = k + l
12:               si no
13:                  A_it = k
14:                  A_jt = l
15:               fin si
16:            fin si
17:            t = t + 1
18:         fin para
19:      fin para
20:   fin para
21: fin para
22: devolver T, A, HQ
```

La función de *cruce* utilizada es del tipo uniforme, para lo cual se genera una cadena aleatoria de unos y ceros del tamaño del cromosoma para escoger el padre del cual se toma cada uno de los genes para crear cada hijo. El criterio de *selección* de los padres es por torneo, donde se seleccionan aleatoriamente dos pares de individuos y se enfrentan entre sí para obtener el mejor de cada par para ser padres de la siguiente generación. Estas funciones se muestran en el algoritmo 16.

La función de *mutación* se define con una probabilidad de mutación P_m. Cada gen de cada individuo de la población tiene asociada una probabilidad P_m de sufrir una mutación. Para llevar a cabo dicha mutación se suma o resta una cantidad

Algoritmo 16 Algoritmo Genético - función de cruce.

Entrada: Pop_{old}, N_{ind}, N_{elite}

1: $N_{hijos} = 0$
2: **mientras** $N_{hijos} + N_{elite} < N_{ind}$ **hacer**
3: $r_1, r_2, r_3, r_4 = rand() \cdot N_{ind}$ (* generar 4 números aleatorios en $[1, N_{ind}]$ *)
4: $patron = rand[cromtam]$ (* patrón aleatorio de 1s y 0s tamaño cromosoma *)
5: **si** $Pop_{old}[r_1](fitness) > Pop_{old}[r_2](fitness)$ **entonces**
6: $padre_1 = Pop_{old}[r_1]$
7: **si no**
8: $padre_1 = Pop_{old}[r_2]$
9: **fin si**
10: **si** $Pop_{old}[r_3](fitness) > Pop_{old}[r_4](fitness)$ **entonces**
11: $padre_2 = Pop_{old}[r_3]$
12: **si no**
13: $padre_2 = Pop_{old}[r_4]$
14: **fin si**
15: $hijo_1 = patron \cdot padre_1 + (1 - patron) \cdot padre_2$
16: $hijo_2 = (1 - patron) \cdot padre_1 + patron \cdot padre_2$
17: agregar $hijo_1$ e $hijo_2$ a Pop_{new}
18: $N_{hijos} = N_{hijos} + 2$
19: **fin mientras**
20: **devolver** Pop_{new}

aleatoria entre cero y la mitad del valor calculado C_{mut} al gen mutado. Cuando un elemento x_j de $\mathbf{X}$ toma un valor relativamente pequeño o un valor negativo (i.e. $x_j \leq 10000$), entonces se hace $x_j = 0$, ya que no es rentable utilizar patrones para procesar cantidades pequeñas de cartón, como por ejemplo, procesar 100 metros. El proceso se ilustra en el algoritmo 17.

Algoritmo 17 Algoritmo Genético - función de mutación.

Entrada: Pop_{old}, P_m, C_{mut}, N_{ind}, $cromtam$

$N_{hijos} = 0$
para $i = 1$ hasta N_{ind} **hacer**
  para $j = 1$ hasta $cromtam$ **hacer**
    $p = rand()$
    si $p < P_m$ **entonces**
      $r = rand(-0.5, 0.5) \cdot C_{mut}$
      $Pop_{new}[i, j] = Pop_{old}[i, j] + r$
    si no
      $Pop_{new}[i, j] = Pop_{old}[i, j]$
    fin si
    si $Pop_{new}[i, j] < 10000$ **entonces**
      $Pop_{new}[i, j] = 0$
    fin si
  fin para
fin para
devolver Pop_{new}

La *función objetivo* utilizada en esta primera aproximación es la definida en la ecuación 4.10. El algoritmo 18 muestra la función para calcular el fitness de cada individuo, de acuerdo con la función objetivo.

Algoritmo 18 Algoritmo Genético - función de fitness 1.

Entrada: Pop_{old}, A, T, N_{ind}, $cromtam$, $TotPedidos$

para $i = 1$ hasta N_{ind} **hacer**
  $cubr = suma(|HQ - A \times Pop_{old}[i]|)$ (* cubrimiento de pedidos *)
  $desp = T \times Pop_{old}[i]$ (* castigo por desperdicio *)
  $used = \text{nnz}(Pop_{old}[i])$ (* castigo por patrones usados *)
  $fit[i] = cubr + desp + used$
fin para
devolver fit

Ejemplo

Para ilustrar el comportamiento del método hemos utilizado el mismo ejemplo anterior con los datos que se presentan en la tabla 4.1. El ancho mínimo es de 192cm y el ancho máximo es de 197cm. Los patrones que cumplen esta restricción son los mismos de la tabla 4.5.

Con esta información se pueden construir la matriz **A** y los vectores **T** y **HQ**. La matriz **A** tiene la misma forma que la del ejemplo anterior de la programación lineal, como se aprecia en las restricciones de la ecuación 4.9. El vector **T** corresponde al desperdicio de cada uno de los patrones y el vector **HQ** a los centímetros lineales que se deben procesar de cada orden. Además se calcula el parámetro $C_{mut} = \overline{HQ} \cdot n/m = 436250 \cdot 5/4 = 545312$. Se genera una población inicial, por ejemplo, como la que se muestra en la tabla 4.8.

Tabla 4.8: Población inicial del AG para el ejemplo con 5 pedidos.

Individuo	x_1	x_2	x_3	x_4	Fitness
1	211000	485000	109000	246000	11058000
2	160000	390000	220000	25000	5083000
3	519000	381000	322000	401000	17296000
4	62000	287000	367000	446000	12754000
5	530000	231000	242000	447000	16598000
6	212000	358000	429000	338000	10796000
7	378000	493000	471000	398000	17609000
8	223000	220000	153000	340000	9635000
9	230000	215000	123000	110000	5019000
10	516000	443000	201000	509000	21197000

Después de ejecutar el algoritmo, la solución óptima que obtenemos es la que se muestra en la tabla 4.9. Como podemos apreciar, los 4 patrones generados son utilizados. En este caso, la solución óptima es similar a la solución obtenida mediante la Programación Lineal. Es bueno recordar además, que debido a la naturaleza probabilística de los Algoritmos Genéticos, en cada ejecución se pueden obtener soluciones diferentes, en especial para problemas de mayor tamaño.

En la tabla 4.10 se muestra el cumplimiento de las órdenes que se obtiene mediante el uso de la solución óptima encontrada. Se puede apreciar que en este caso no todos los pedidos se cumplen dentro del margen del 10% permitido, ya que del pedido número 4 se produce sólo un 71% de las unidades ordenadas, dejando 867 unidades pendientes para una posterior programación, y del pedido 3 se produce poco más del 110%, generando un excedente de 16 cajas de este tipo. En este caso el porcentaje total programado fue un 104% de la cantidad ordenada. En total se utilizaron 4

Tabla 4.9: Primera solución por AG para el ejemplo con 5 pedidos.

Patrón	T	X
1	1,4	167250
2	0,5	244580
3	2,7	268060
4	4,6	120230

patrones y el desperdicio de material es del 1.15%, contando como desperdicio las cajas producidas en exceso, o del 1.04% contando solamente el desperdicio normal. El fitness de la mejor solución es 2384000.

Tabla 4.10: Cumplimiento de las órdenes con la primera solución por AG.

Pedido	Ordenado	Producido	Pendiente	Exceso
1	4000	4390	0	0
2	6000	6592	0	0
3	2500	2766	0	16
4	3000	2133	867	0
5	5000	5500	0	0
Total	**20500**	**21381**	**867**	**16**

4.3.2 Segunda formulación - Método AG2

Para mejorar la aproximación anterior, se plantea una nueva función objetivo donde se tienen en cuenta todas las variables que nos permiten optimizar el proceso y mantener congruencia en las unidades utilizadas en los diferentes términos de la función. Como se ha dicho anteriormente, la función objetivo debería incluir las variables que midan el desperdicio, el cumplimiento de los pedidos y los cambios de cuchillas. Además, como no restringimos la cantidad máxima procesada de cada pedido, también se debe castigar el desperdicio por exceso de unidades. En esta aproximación utilizamos la siguiente función objetivo, función de adaptación, o fitness value:

$$\frac{\sum \min(\mathbf{A}_{i*}\mathbf{X}, HQ_i) \cdot W_i}{\sum HQ_i \cdot W_i} - \frac{\sum T_j X_j + \sum \max((\mathbf{A}_{i*}\mathbf{X} - HQ_i) \cdot W_i, 0) + k_t \cdot nnz(\mathbf{X}) \cdot W_i}{\sum \mathbf{A}_{i*}\mathbf{X} \cdot W_i}$$

donde

$\sum \min(\mathbf{A}_{i*} \times \mathbf{X}, HQ_i) \cdot W_i$ Es el total de cartón útil, en centímetros cuadrados que se procesa, sin contar el exceso.

$\sum HQ_i \cdot W_i$ Es el total de cartón en centímetros cuadrados que se debe procesar para cumplir con todas las órdenes.

$\sum T_j X_j$ Es el total de centímetros cuadrados en las tiras de desperdicio producidas al procesar los patrones seleccionados.

$\sum \max((\mathbf{A}_{i*} \times \mathbf{X} - HQ_i) \cdot W_i, 0)$ Es el total de desperdicio por exceso de producción en centímetros cuadrados.

$nnz(\mathbf{X}) \cdot W_i$ Es la cantidad de cartón en centímetros cuadrados que podría procesarse durante los cambios de cuchillas según los patrones.

$k_t = \frac{t_p}{t_{cm}}$ Es la cantidad de centímetros lineales que se podrían recortar durante cada cambio de patrón en la máquina. Corresponde al tiempo que tarda un cambio de cuchillas, dividido sobre el tiempo que se tarda la máquina en recortar un centímetro de material.

Algoritmo 19 Algoritmo Genético - función de fitness 2.

Entrada: Pop_{old}, A, T, N_{ind}, $cromtam$, $TotPedidos$

1: **para** $i = 1$ hasta N_{ind} **hacer**
2: $prog = suma(\min((A \times Pop_{old}[i]), HQ) \cdot W)$ (* total programado *)
3: $desp = T \times Pop_{old}[i]$ (* castigo por desperdicio *)
4: $exce = suma(\max(A \times Pop_{old}[i] - HQ, 0) \cdot W)$ (* castigo por exceso *)
5: $used = 1000\text{nnz}(Pop_{old}[i]) \cdot W_{min}$ (* castigo por patrones usados *)
6: $fit[i] = prog/TotPedidos - (desp + exce + used)/prog$
7: **fin para**
8: **devolver** $-fit$

Ejemplo

Nuevamente se utiliza el ejemplo que se muestra en la tabla 4.1 para ilustrar el comportamiento de este nuevo método. Los patrones utilizados son los mismos presentados en la tabla 4.5. Como se ha dicho, el único cambio con respecto a la formulación anterior se encuentra en la función objetivo o fitness de los individuos. La tabla 4.11 muestra los valores de fitness para la misma población inicial del ejemplo anterior.

Después de ejecutar el algoritmo, la solución óptima que obtenemos es la mostrada en la tabla 4.12. Como podemos apreciar, los 4 patrones generados son utilizados. Nuevamente la solución óptima es similar a la solución que se obtiene mediante la Programación Lineal y a la que se obtiene con la primera formulación de AG.

En la tabla 4.13 se muestra el cumplimiento de las órdenes con el uso de la solución óptima encontrada. Nuevamente, no todos los pedidos se cumplen dentro del margen

Tabla 4.11: Población inicial del AG para el ejemplo con 5 pedidos.

Individuo	x_1	x_2	x_3	x_4	Fitness
1	211000	485000	109000	246000	5,64
2	160000	390000	220000	25000	0,82
3	519000	381000	322000	401000	8,74
4	62000	287000	367000	446000	4,85
5	530000	231000	242000	447000	8,06
6	212000	358000	429000	338000	4,45
7	378000	493000	471000	398000	9,12
8	223000	220000	153000	340000	3,23
9	230000	215000	123000	110000	0,31
10	516000	443000	201000	509000	11,87

Tabla 4.12: Segunda solución por AG para el ejemplo con 5 pedidos.

Patrón	T	X
1	1,4	166870
2	0,5	244610
3	2,7	266460
4	4,6	120020

del 10% permitido, ya que del pedido número 4 se produce sólo un 70% de las unidades ordenadas, dejando 880 unidades pendientes para una posterior programación. En este caso ninguno de los pedidos se ha producido en exceso. El porcentaje total programado fue un 104% de la cantidad ordenada. En total se utilizaron 4 patrones y el desperdicio de material es del 1.05%. El fitness de la mejor solución es -0.9175.

Tabla 4.13: Cumplimiento de las órdenes con la segunda solución por AG.

Pedido	Ordenado	Producido	Pendiente	Exceso
1	4000	4380	0	0
2	6000	6593	0	0
3	2500	2750	0	0
4	3000	2120	880	0
5	5000	5494	0	0
Total	**20500**	**21337**	**880**	**0**

4.3.3 Formulación de un método híbrido

Después de hacer algunas pruebas con los métodos anteriores y analizar las fortalezas y debilidades de cada uno de ellos, se ha formulado un nuevo método híbrido, que combine el algoritmo genético y la programación lineal para mejorar el desempeño general del algoritmo. En este caso utilizamos como base el método AG2, agregando a la población inicial algunos individuos generados por medio de la programación lineal del método PL2 con algunas variaciones pequeñas para lograr conseguir 4 individuos diferentes que hacen parte ahora de la población inicial, originalmente aleatoria. Si tenemos en cuenta que el algoritmo genético utiliza la función de elitismo, estos individuos tienen una gran posibilidad de sobrevivir durante las primeras generaciones, y ayudar a mejorar la solución final. Además se utiliza un nuevo término en la función objetivo con el que se busca reducir el número de pedidos que son programados pero quedan incompletos. Es decir, se castiga por cada pedido que haya sido producido en un número de unidades mayor que cero y menor que el 90% mínimo requerido. Esto con el fin de aumentar el porcentaje de cumplimiento de los pedidos, a la vez que no se dejan pedidos incompletos faltando una cantidad mínima por producir, ya que esto dificulta la futura programación de dicho pedido y el proceso en general de producción. En el algoritmo 20 se presenta la nueva función de fitness utilizada en este método. Aparte de la función de fitness y la generación de la población inicial, los demás parámetros del algoritmo fueron sintonizados también durante las pruebas para lograr el mejor desempeño posible en la mayoría de los casos.

Los resultados obtenidos con este método son bastante satisfactorios. En la tabla 4.14 se muestra el cumplimiento de las órdenes que se obtiene mediante el uso de la

Algoritmo 20 Algoritmo Híbrido - función de fitness.

Entrada: Pop_{old}, A, T, N_{ind}, $cromtam$, $TotPedidos$

1: **para** $i = 1$ hasta N_{ind} **hacer**
2: $prog = suma(\min((A \times Pop_{old}[i]), HQ) \cdot W)$ (* total programado *)
3: $desp = T \times Pop_{old}[i]$ (* castigo por desperdicio *)
4: $exce = suma(\max(A \times Pop_{old}[i] - HQ, 0) \cdot W)$ (* castigo por exceso *)
5: $used = 1000\text{nnz}(Pop_{old}[i]) \cdot W_{min}$ (* castigo por patrones usados *)
6: $comp = C_c \sum(0 < A \times Pop_{old}[i] < 0.9HQ)$ (* castigo pedidos incompletos *)
7: $fit[i] = prog/TotPedidos - (desp + exce + used + comp)/prog$
8: **fin para**
9: **devolver** $-fit$

solución óptima encontrada para el ejemplo planteado anteriormente en la tabla 4.1. Se puede apreciar que la solución en este caso es igual a la que se obtuvo anteriormente con el método PL2. Sin embargo, en las diferentes pruebas realizadas, el nuevo método híbrido supera tanto a los métodos de programación lineal como a los algoritmos genéticos.

Tabla 4.14: Cumplimiento de las órdenes con el método híbrido.

Pedido	Ordenado	Producido	Pendiente
1	4000	4400	0
2	6000	6600	0
3	2500	2750	0
4	3000	2120	880
5	5000	5500	0
Total	**20500**	**21370**	**880**

Capítulo 5

Experimentación

Para analizar los resultados que se pueden obtener con los métodos explicados en el capítulo anterior se realizaron una serie de pruebas con diferentes conjuntos de datos y diferentes condiciones, de acuerdo con las situaciones que se podrían presentar en un contexto real. En este capítulo se presentan los resultados obtenidos en dichas pruebas y se hace un análisis de los resultados obtenidos.

5.1 Pruebas

Para realizar las diferentes pruebas y el subsecuente análisis comparativo se utilizaron datos reales de una empresa para 120 pedidos. La información de los pedidos puede observarse en la tabla A.1 del apéndice A. Con estos datos se llevaron a cabo 4 pruebas diferentes: con 20, 40, 80 y 120 pedidos. Para las pruebas con 20, 40 y 80 pedidos, se formaron diferentes grupos de pedidos con estas cantidades y se utilizaron los diferentes métodos para hallar la mejor combinación. Para la prueba con 120 pedidos se realizó la programación simulando el ingreso de pedidos a la empresa durante 5 días, es decir, que en cada corrida se utilizan los pedidos que corresponden a ese día más los pedidos pendientes de los días anteriores. En esta prueba se programan grupos de entre 15 y 30 pedidos. En las tablas A.2 a A.6 se muestran los resultados de cada prueba individual con los diferentes métodos. En las tablas 5.1 a 5.4 se muestra un promedio de los resultados obtenidos con cada método. En cada tabla se muestra el porcentaje de programación de los pedidos, el porcentaje de desperdicio de material, el total de patrones utilizados y el porcentaje de los pedidos completados en las diferentes pruebas.

Los resultados para el primer método de programación lineal (PL1) se muestran en la tabla 5.1. Como se explicó anteriormente, al utilizar este método siempre se cumple con el 100% de los pedidos, cumpliendo en general con un 95% a 97% del total

de unidades ordenadas. El desperdicio producido se reduce al utilizar este método con una mayor cantidad de pedidos, gracias al mayor número de patrones posibles que se pueden generar. El número de patrones utilizados es bastante alto, haciendo un uso promedio de entre 4 y 5 patrones para completar cada pedido.

Tabla 5.1: Resultados de las pruebas para el método PL1

	Prueba 20	Prueba 40	Prueba 80	Prueba 120
Programación	95,11%	96,48%	96,31%	95,39%
Desperdicio	4,08%	1,41%	0,70%	2,53%
Patrones	92	168	444	91
Completados	100%	100%	100%	100%

Los resultados para el segundo método de programación lineal (PL2) se muestran en la tabla 5.2. Se puede observar que el comportamiento del método varía de acuerdo con el número de pedidos en la programación. En general, y teniendo en cuenta que la prueba de 120 pedidos en realidad corresponde a pruebas entre 20 y 30 pedidos, a mayor número de pedidos, menor es el desperdicio producido y mayor es el porcentaje programado. También el número de patrones utilizados crece cuando aumenta el número de pedidos, debido a la mayor cantidad de combinaciones que pueden lograrse dentro de los límites de desperdicio establecidos.

Tabla 5.2: Resultados de las pruebas para el método PL2

	Prueba 20	Prueba 40	Prueba 80	Prueba 120
Programación	73,11%	99,24%	107,92%	88,21%
Desperdicio	1,36%	1,45%	1,40%	1,35%
Patrones	23	86	252	40
Completados	68%	86%	98%	75%

Los resultados para el primer método de algoritmos genéticos (AG1) se muestran en la tabla 5.3. Se puede observar que el desperdicio generado por las unidades producidas en exceso es bastante elevado. El porcentaje de programación es alto, pero el porcentaje de cumplimiento de los pedidos nos da a entender que dichos niveles de programación se deben en parte al exceso en la producción.

Los resultados para el segundo método de algoritmos genéticos (AG2) se muestran en la tabla 5.4. Claramente el desperdicio generado por las unidades producidas en exceso es mucho menor que en el método anterior. En general, este método tiene un comportamiento muy estable, independiente del número de pedidos.

Los resultados para el método híbrido (HIB) se muestran en la tabla 5.5. A pesar que el desperdicio generado es un poco mayor al que se genera al utilizar los métodos

Tabla 5.3: Resultados de las pruebas para el método AG1

	Prueba 20	Prueba 40	Prueba 80	Prueba 120
Programación	87,03%	107,25%	107,77%	102,67%
Desperdicio Lineal	1,15%	0,97%	0,92%	1,18%
Desperdicio por Exceso	5,64%	3,85%	3,39%	5,85%
Desperdicio Total	6,78%	4,82%	4,32%	7,03%
Patrones	20	62	183	33
Completados	76%	90%	85%	79%

Tabla 5.4: Resultados de las pruebas para el método AG2

	Prueba 20	Prueba 40	Prueba 80	Prueba 120
Programación	67,86%	86,03%	87,39%	81,19%
Desperdicio Lineal	1,28%	1,24%	1,17%	1,37%
Desperdicio por Exceso	0,01%	0,13%	0,42%	0,06%
Desperdicio Total	1,28%	1,38%	1,58%	1,43%
Patrones	17	54	160	32
Completados	55%	60%	57%	61%

PL2 y AG2, el porcentaje de cumplimiento es mayor y el número de patrones utilizados es bastante bajo. En general, con este método se cumple el propósito de aprovechar las ventajas mostradas por los demás métodos y lograr construir un método mejor.

Tabla 5.5: Resultados de las pruebas para el método HIB

	Prueba 20	Prueba 40	Prueba 80	Prueba 120
Programación	85,19%	101,62%	98,90%	91,61%
Desperdicio Lineal	2,11%	2,28%	1,17%	1,68%
Desperdicio por Exceso	0,03%	0,17%	0,28%	0,13%
Desperdicio Total	2,14%	2,45%	1,45%	1,81%
Patrones	18	60	102	29
Completados	81%	91%	86%	80%

5.1.1 Análisis de resultados

A continuación se hace un análisis de los resultados más importantes que arrojan estas pruebas, comparando los cuatro métodos de acuerdo a los parámetros evaluados, que son el porcentaje de programación, el porcentaje de desperdicio, el número de patrones utilizados y el porcentaje de pedidos completados.

Porcentaje de programación

El porcentaje de programación hace referencia al total de centímetros cuadrados producidos sobre el total de centímetros cuadrados que se deberían producir, según los pedidos de los clientes.

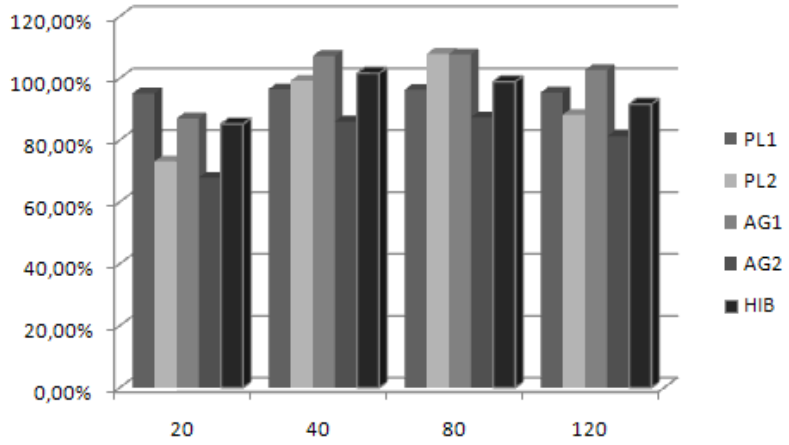


Figura 5.1: Porcentaje de Programación

En la figura 5.1 se observa que el método PL1 tiene un buen desempeño en este aspecto, cumpliendo siempre entre el 90% y el 100% de los pedidos. Los demás métodos aumentan el porcentaje de programación cuando son utilizados con un mayor número de pedidos. Es así como para 80 pedidos es mayor el porcentaje de programación alcanzado que para solamente 20 pedidos. Este comportamiento es lógico si se tiene en cuenta que para estos métodos se utiliza únicamente un subconjunto de los patrones que pueden ser utilizados en el método PL1, aquellos que producen un menor desperdicio. El algoritmo AG1 parece tener en general un alto porcentaje de programación, aunque en este porcentaje se incluyen los centímetros cuadrados correspondientes a la producción en exceso, así que en realidad no toda la producción que se muestra para este método será producción útil. El algoritmo PL2 y el algoritmo híbrido tienen un desempeño muy bueno. Por otro lado, el algoritmo AG2 tiene el desempeño más bajo. Esto se debe en gran parte a que se le da un peso importante a minimizar el número de patrones utilizados, pero puede ser ajustado de diferentes maneras, según sea conveniente.

Porcentaje de desperdicio total

El porcentaje de desperdicio corresponde al total de centímetros cuadrados de desperdicio generados sobre el total de centímetros cuadrados de material útil producidos. En el desperdicio total se tienen en cuenta tanto el desperdicio lineal que genera cada patron al ser utilizado, como el exceso de producción por unidades que superan el 110% de la orden.

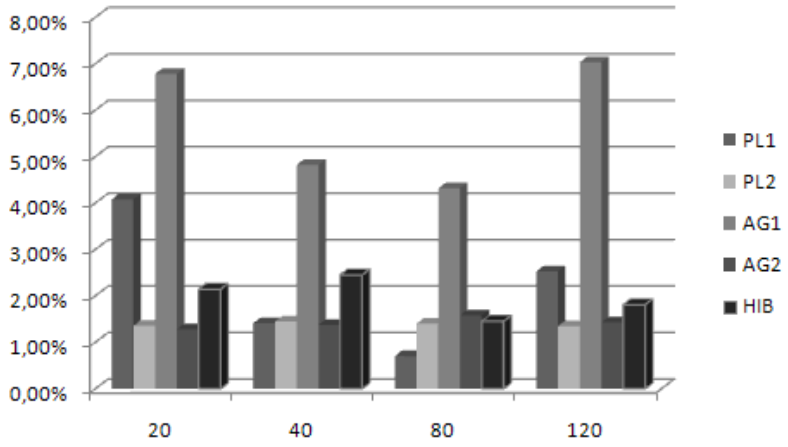


Figura 5.2: Desperdicio Total

En la figura 5.2 se observa que el método AG1 tiene un desperdicio bastante elevado, ocasionado principalmente por la producción en exceso que este método genera. El algoritmo PL1 también tiene un nivel elevado de desperdicio, aunque en este caso el desperdicio se reduce cuando el número de pedidos aumenta, consecuencia del mayor número de patrones posibles que se pueden utilizar. Los métodos PL2 y AG2 tienen un excelente desempeño en este aspecto, con un porcentaje de desperdicio muy estable y que no supera el 1.6% en ningún caso. En general, el algoritmo híbrido genera un desperdicio mayor a estos dos métodos, pero aún dentro de los límites aceptables.

Número de patrones utilizados

Como ya se ha señalado anteriormente, es importante minimizar también el número de patrones utilizados, puesto que cada vez que hay un cambio de patrón, la máquina debe detenerse por unos minutos, por lo que entre más paradas se hagan, menos tiempo de producción se tiene.

Analizando la figura 5.3, se observa que el método PL1 utiliza un número de

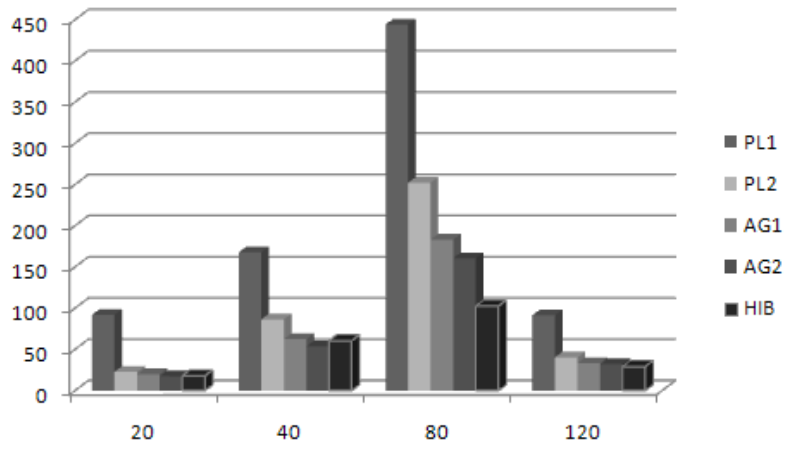


Figura 5.3: Número de patrones de corte utilizados

patrones bastante elevado. En este método se utilizan 3 o más patrones por cada pedido. Los métodos que menos patrones utilizan en todos los casos son el AG2 y el algoritmo híbrido, lo que demuestra la importancia que se le da a este aspecto de optimización en estos métodos.

Porcentaje de pedidos completados

El porcentaje de pedidos completados hace referencia al número de pedidos que fueron procesados en más del 90% de las unidades ordenadas, sobre el número total de pedidos.

En la figura 5.4 se observa que el método PL1 completa siempre el 100% de los pedidos como se esperaba, debido a la formulación del método. El método AG2 completa entre un 50% y un 60% de los pedidos en cada prueba, lo cual es bastante bajo. Los métodos PL2 y AG1 tienen un comportamiento regular en este aspecto, superados también por el método híbrido, con el que se completan normalmente más del 80% de los pedidos.

Ponderación

Para poder comparar los cinco métodos planteados y poder decidir cuál de los métodos es más efectivo en la solución de este problema, es necesario analizar los resultados anteriores dándole un peso o nivel de importancia a cada uno de los parámetros evaluados. A cada método se le da una calificación de 1 a 10 de acuerdo a qué tan buen desempeño obtuvo en cada aspecto evaluado. Estas calificaciones se multiplican

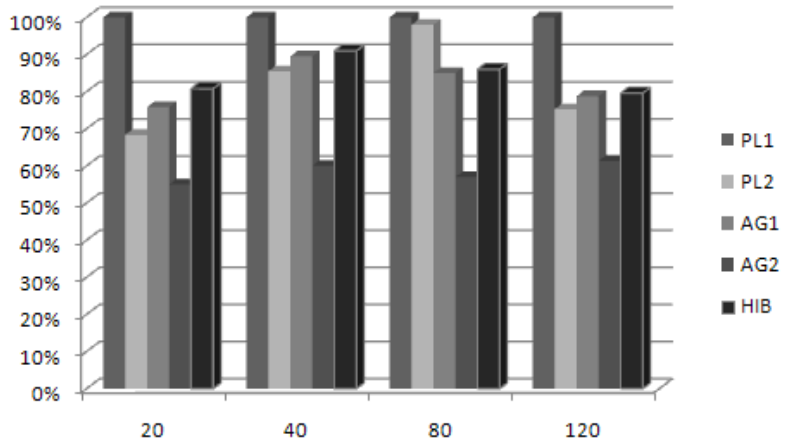


Figura 5.4: Porcentaje de pedidos completados

por los pesos correspondientes para obtener el puntaje total de cada método. El máximo puntaje que se puede obtener es 1000.

Como sabemos, lo más importante es minimizar el desperdicio producido en el proceso, por lo cual el mayor peso lo tiene el *porcentaje de desperdicio total*, con un 45% del puntaje total. El 55% restante se divide entre el *número de patrones utilizados* (30%) y el porcentaje total programado, con un 15% para el *porcentaje de pedidos completados* y un 10% para el *porcentaje de programación*. En la tabla 5.6 se muestra la ponderación de los resultados obtenidos, según la calificación de 1 a 10 que les hemos dado.

Tabla 5.6: Resultados ponderados para los 4 métodos

	Peso	PL1	PL2	AG1	AG2	HIB
Programación	10	9	9	7	5	9
Desperdicio	45	4	10	1	10	9
Patrones	30	1	6	8	10	10
Completados	15	10	7	7	4	8
TOTAL		**450**	**825**	**460**	**860**	**915**

Según nuestra calificación, el método que mejor puntaje obtiene es el algoritmo híbrido, con un total de 915 puntos. Este método tiene una calificación de 10 puntos en el número de patrones utilizados, 9 puntos en el porcentaje de desperdicio y de programación, y 8 puntos en el porcentaje de pedidos completados. En segundo lugar se encuentra el método AG2, con una calificación de 860 puntos. Este método obtuvo

10 puntos en el número de patrones utilizados y el porcentaje de desperdicio total. Un aspecto negativo de este método es que cuenta con el promedio más bajo en cuanto al porcentaje de programación. En tercer lugar se encuentra el método PL2, con un total de 825 puntos. El método PL2 obtuvo una calificación de 10 puntos en el ítem de desperdicio y de 9 puntos en cuanto al porcentaje de programación. El punto débil de este método es el elevado número de patrones que utiliza.

5.1.2 Software

Para facilitar el uso del método propuesto al usuario final, se ha creado una interfaz de usuario bastante sencilla, que permite controlar los diferentes parámetros del algoritmo, ingresar los datos de las órdenes de los clientes desde archivos de Excel y finalmente observar los resultados obtenidos.

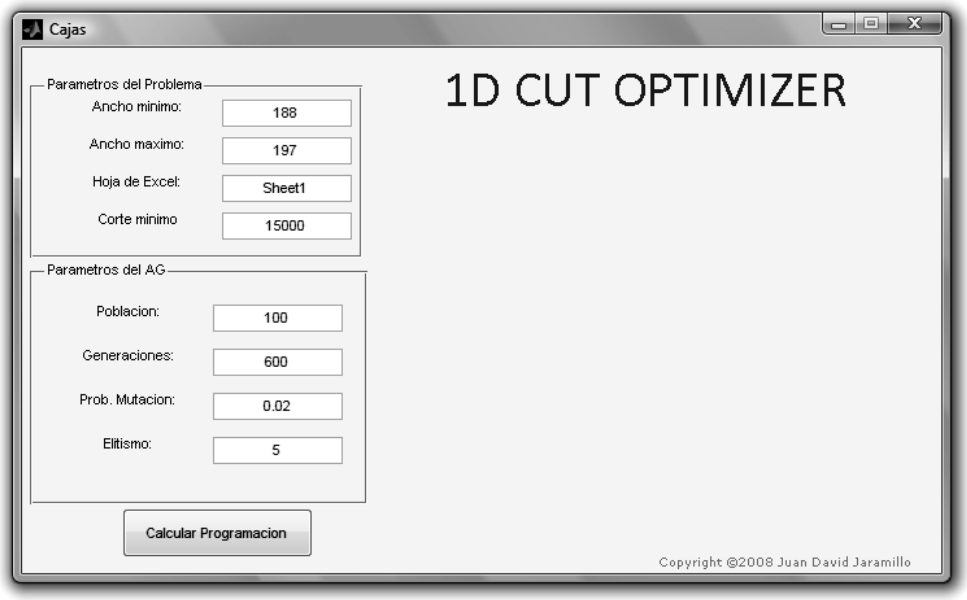


Figura 5.5: Vista inicial del programa

La figura 5.5 muestra la ventana principal del programa. En esta ventana se pueden especificar los valores de algunos parámetros del problema y los parámetros propios del algoritmo genético. Los parámetros ajustables del problema son el ancho máximo de una lámina y el ancho mínimo que se permite utilizar, así como la longitud mínima que debe tener cada corte. Para sintonizar el algoritmo genético se cuenta con los parámetros del tamaño de la población, número máximo de generaciones, probabilidad de mutación y porcentaje de elitismo. Todos los parámetros están ajustados

a un valor por defecto seleccionado a partir de los resultados obtenidos en numerosas pruebas.

Al hacer click en el botón *Calcular Programación*, se abre el diálogo para seleccionar un archivo, como se muestra en la figura 5.6. Únicamente pueden seleccionarse archivos con extensión xls. Este archivo debe contener cuatro columnas de datos: identificador del pedido, ancho de la caja, largo de la caja y cantidad de cajas ordenadas. Después de ejecutarse el algoritmo, en este mismo archivo se agregan las columnas del número de cajas programadas y el número de cajas pendientes de programación.

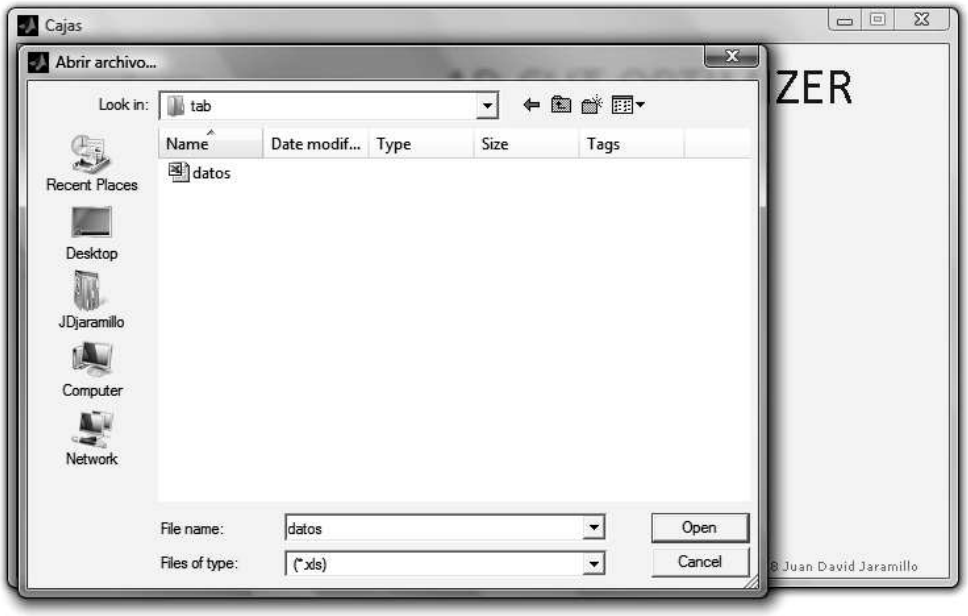


Figura 5.6: Vista: seleccionar archivo de datos

Después de hacer click en *Abrir*, el programa se ejecuta automáticamente. Mientras el algoritmo está en ejecución, una barra muestra el porcentaje completado, para tener una idea del tiempo de ejecución restante, como se muestra en la figura 5.7.

Luego de ejecutarse el algoritmo se presenta un resumen de los resultados obtenidos, como se aprecia en la figura 5.8. En este cuadro se puede observar el número de pedidos completados y el total de pedidos, el número de unidades (cajas) producidas y el total ordenadas, el porcentaje de desperdicio producido, el número de patrones utilizados, la longitud promedio de cada corte y la longitud total de corte entre todos los patrones utilizados.

Haciendo uso de los botones ubicados en la parte inferior, se puede acceder a diferentes vistas de la solución. El primer botón muestra la vista de cumplimiento de

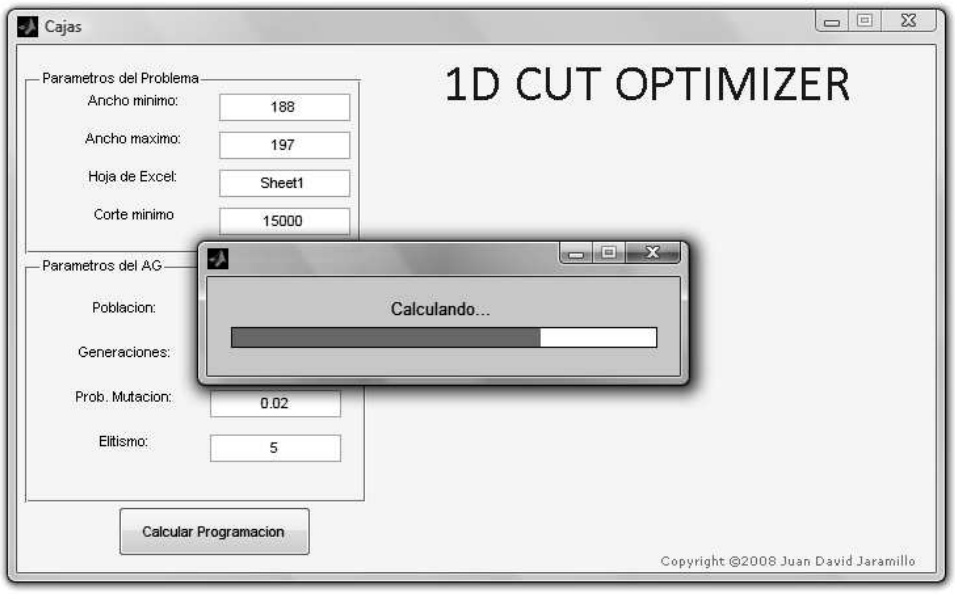


Figura 5.7: Vista: procesamiento de datos

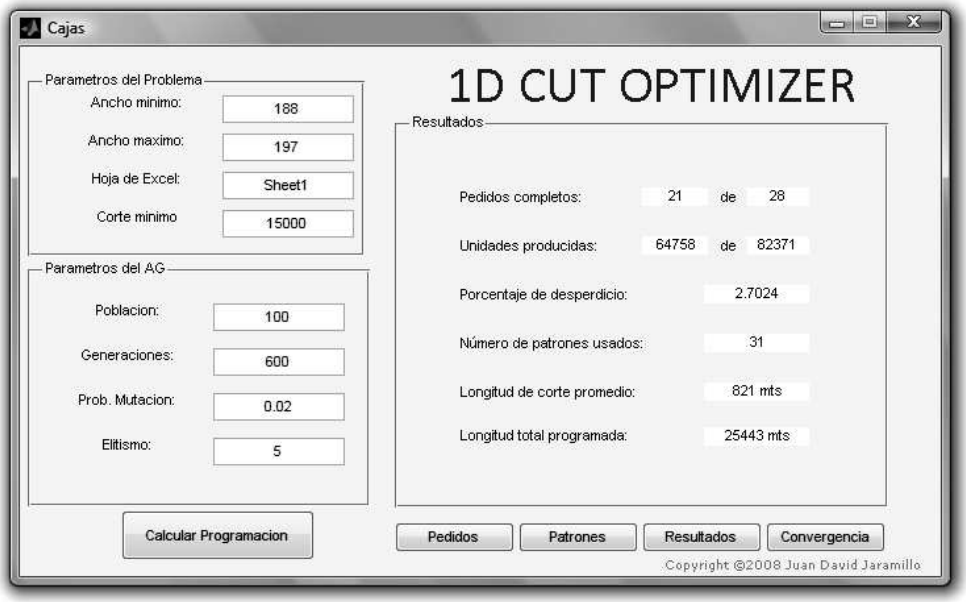


Figura 5.8: Vista: resultados obtenidos

los pedidos, que se presenta en la figura 5.9. En la tabla se muestra la información de cada pedido (id, ancho, largo, cantidad), además del número de unidades producidas y el número de unidades pendientes por producir. Al hacer click en *Abrir Archivo*, se puede acceder al archivo de Excel que cuenta con la misma información, que se muestra en la figura 5.10.

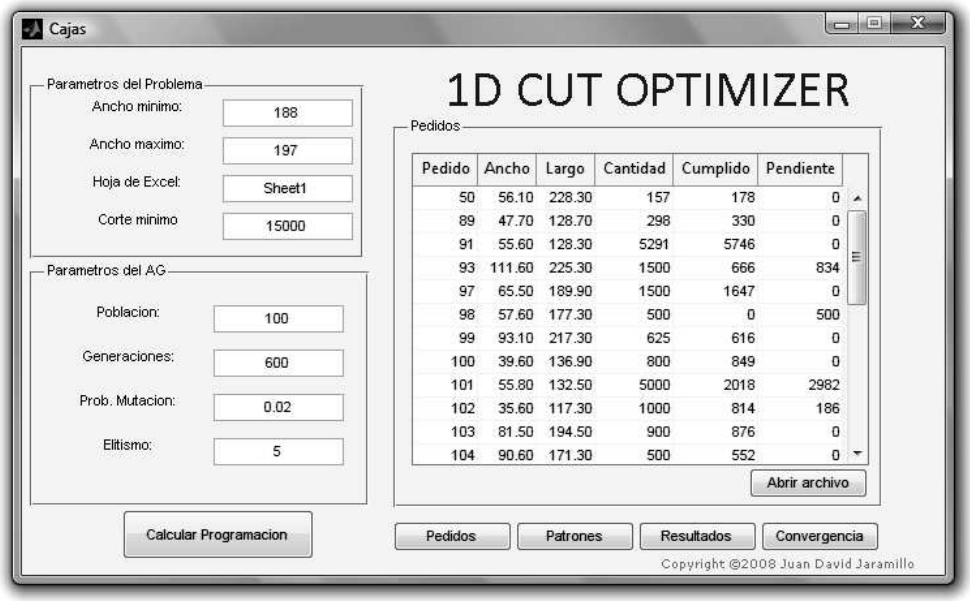
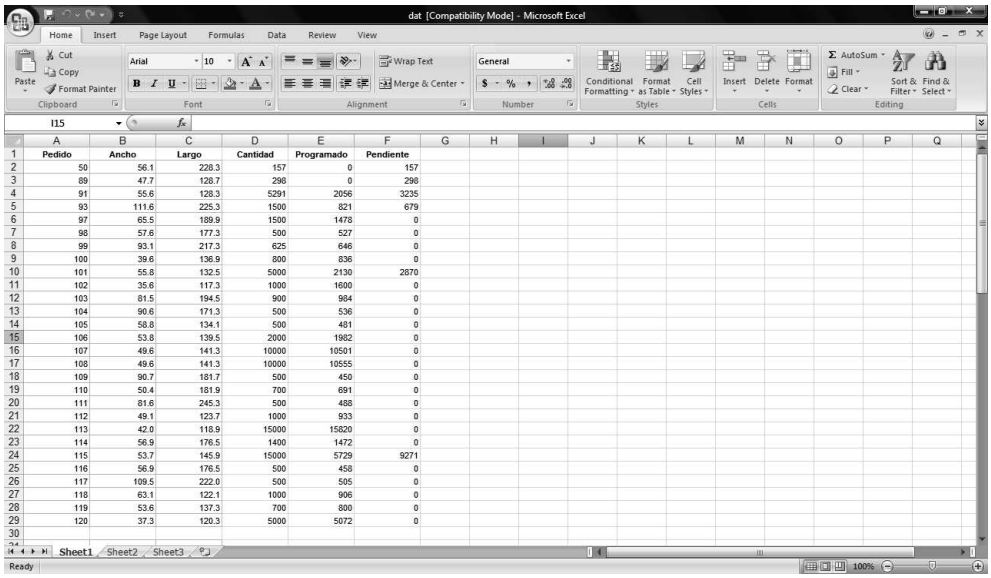


Figura 5.9: Vista: cumplimiento de los pedidos

El segundo botón muestra la vista de los patrones generados, que se presenta en la figura 5.11. En la tabla se muestra la información de cada patrón, es decir los identificadores de ambos pedidos utilizados, la cantidad de unidades de cada uno que se utilizan horizontalmente y la longitud total que debe recortarse con dicho patrón. Al hacer click en *Abrir Archivo*, se puede acceder al archivo de Excel que cuenta con la misma información, que se muestra en la figura 5.12.

Finalmente, con el botón *Convergencia* se puede acceder a la gráfica de la convergencia del algoritmo genético. Esta gráfica muestra el *fitness promedio* de la población en cada generación.



Pedido	Ancho	Largo	Cantidad	Programado	Pendiente
50	56.1	228.3	157	0	157
89	47.7	128.7	298	0	298
91	55.6	128.3	5291	2056	3235
93	111.6	225.3	1500	821	679
97	65.5	189.9	1500	1478	0
98	57.6	177.3	500	527	0
99	93.1	217.3	625	646	0
100	39.6	136.9	800	836	0
101	55.8	132.5	5000	2130	2870
102	35.6	117.3	1000	1600	0
103	81.5	194.5	900	984	0
104	90.6	171.3	500	536	0
105	58.8	134.1	500	481	0
106	53.8	139.5	2000	1982	0
107	49.6	141.3	10000	10501	0
108	49.6	141.3	10000	10555	0
109	90.7	181.7	500	450	0
110	50.4	181.9	700	691	0
111	81.6	245.3	500	488	0
112	49.1	123.7	1000	933	0
113	42.0	118.9	15000	15820	0
114	56.9	176.5	1400	1472	0
115	53.7	145.9	15000	5729	9271
116	56.9	176.5	500	458	0
117	109.5	222.0	500	505	0
118	63.1	122.1	1000	906	0
119	53.6	137.3	700	800	0
120	37.3	120.3	5000	5072	0

Figura 5.10: Archivo de datos de los pedidos

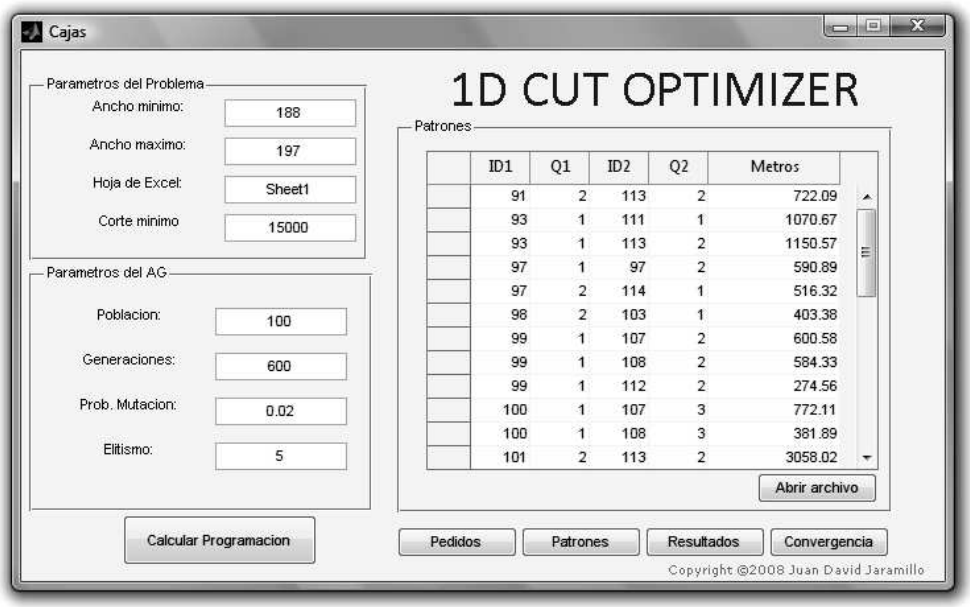


Figura 5.11: Vista: patrones generados

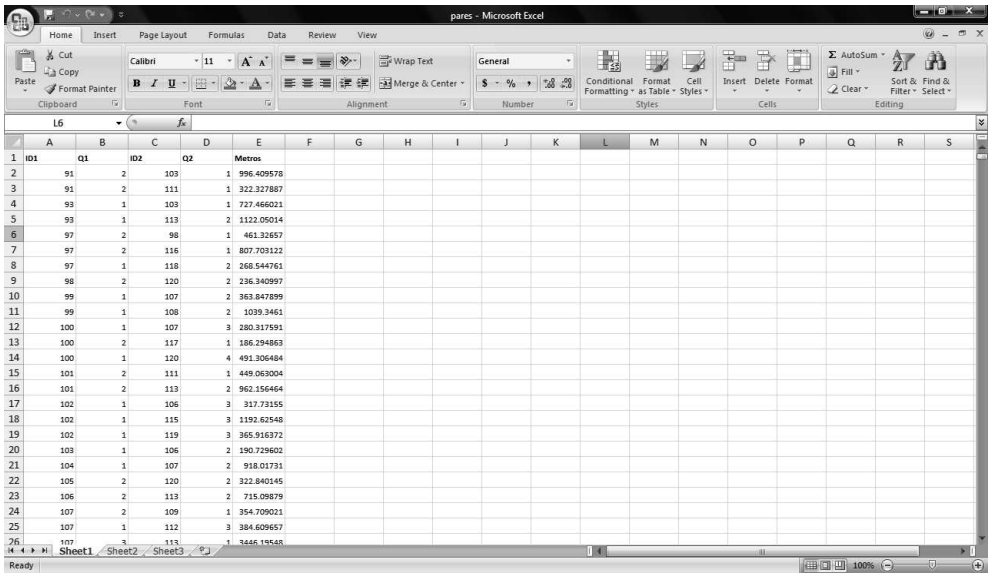


Figura 5.12: Archivo de patrones generados

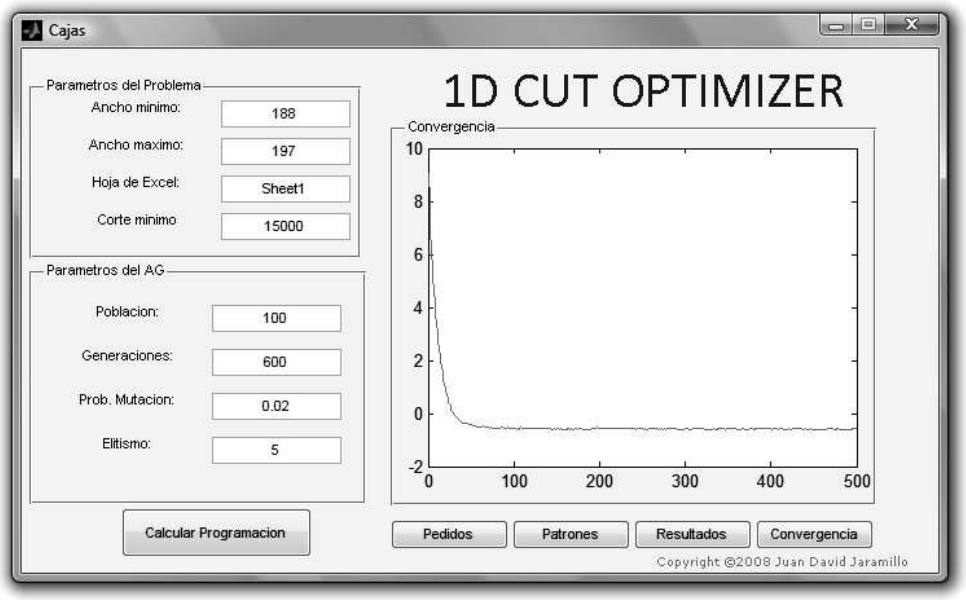


Figura 5.13: Vista: convergencia del algoritmo

Capítulo 6

Conclusiones y trabajo futuro

Los algoritmos genéticos desarrollados en este proyecto son una alternativa que ha superado a los métodos de la programación lineal clásica en cuanto a que logran optimizar en aspectos donde el otro método se queda corto. El modelo utilizado en los algoritmos genéticos es mucho más completo y cercano al problema real que los modelos planteados en la programación lineal, debido a que permiten incluir una mayor cantidad de elementos y condiciones propias del problema. De acuerdo con las pruebas realizadas, el algoritmo genético inicial presenta debilidad en el porcentaje de cumplimiento de los pedidos. Para suplir esta falla, presentamos el algoritmo híbrido, que mejora el desempeño del algoritmo genético inicial al apoyarse en la programación lineal y contar con una función objetivo más completa.

El algoritmo híbrido utiliza datos obtenidos como resultado de la ejecución de los algoritmos de programación lineal para generar parte de la población inicial del algoritmo genético, y cuenta además con un nuevo término en la función de fitness, que permite reducir el número de pedidos incompletos. De acuerdo con nuestras pruebas, en términos generales, el algoritmo híbrido supera a los demás, según el porcentaje de desperdicio, el número de patrones utilizados y el porcentaje de programación de los pedidos. Lo anterior se demuestra con el puntaje obtenido en la ponderación de los resultados que hemos planteado.

La optimización multi-objetivo presenta comúnmente retos importantes debido a que los diferentes objetivos de optimización pueden en general ser contradictorios y difíciles de cuantificar. En nuestro caso hemos logrado una función objetivo para el algoritmo genético que optimice en todos los aspectos importantes este proceso productivo específico, con resultados bastante buenos, a pesar de la aleatoriedad propia del algoritmo genético.

Gracias al planteamiento del modelo que hemos propuesto, el problema puede ser escalado a problemas relacionados con el corte de materiales bajo diferentes condi-

ciones. De forma similar, en problemas de 2 y 3 dimensiones, donde creemos que los algoritmos genéticos y la computación paralela son de gran utilidad, debido a las dificultades que surgen al aumentar el tamaño del problema.

El uso de la computación paralela para mejorar el desempeño de nuestro algoritmo se había planteado inicialmente pensando en resolver el problema con una gran cantidad de datos. Sin embargo, los datos suministrados por las empresas del sector hacen que el uso de la computación paralela no sea adecuado. No obstante, creemos que para modelos relacionados con nuestro problema que utilicen una mayor cantidad de datos, la computación paralela puede ser muy benéfica en cuanto a la reducción de tiempos de cómputo.

La herramienta de software construida facilita el trabajo de un operario y permite crear en pocos segundos la programación de un conjunto de pedidos que tenga la empresa. Esta herramienta es simple y permite ajustar los diferentes parámetros del problema y del algoritmo genético desde la propia interfaz. Además, los archivos de MS Excel utilizados para adquirir los datos y guardar los resultados facilitan su uso.

Como trabajo a futuro, planteamos también la posibilidad de construir un *Sistema Experto* que emule los procedimientos lógicos realizados por un experto en la programación del proceso estudiado y permita dotar de un mayor nivel de inteligencia la herramienta para la solución del problema. Adicionalmente, sería conveniente realizar la comparación con el método de programación lineal multiobjetivo.

A partir de los resultados obtenidos con el desarrollo del presente proyecto, se produjeron dos artículos científicos. El primero, *"Programación Lineal y Algoritmos Genéticos para la Solución del Problema de Corte Unidimensional"* se presentó a la Revista Colombiana de Computación en abril de 2008 y está siendo sometido a evaluación. El segundo, *"Desarrollo de un Método Híbrido Basado en Algoritmos Genéticos y Programación Lineal para la Solución de un Problema de Corte Unidimensional"* será presentado a la Revista de Ingeniería de la Universidad Eafit.

Bibliografía

Antoniou A. y Lu W., 2007. *Practical Optimization, Algorithms and Engineering Applications*. Springer.

Chauhan S., Martel A. y D'amours S., 2006. Roll Assortment Optimization in a Paper Mill: An Integer Programming Approach. *Université Laval, Québec*.

Chong E. y Zak S., 2001. *An Introduction to Optimization*. John Wiley and Sons, Inc.

Coley D., 1999. *An Introduction to Genetic Algorithms for Scientists and Engineers*. World Scientific Publishing Co.

Gass S., 1969. *Linear Programming*. McGraw-Hill.

Geist A., 1994. *PVM: Parallel Virtual Machine*. The MIT Press.

Gendreau M., 2002. *An Introduction to Tabu Search*. `http://www.ifi.uio.no/infheur/Bakgrunn/Intro_to_TS_Gendreau.htm`. Université de Montréal.

Gómez R. y Sicard A., 2001. *Informática teórica. Elementos propedéuticos*. Universidad Eafit.

Guerequeta R. y Vallecillo A., 2000. *Técnicas de Diseño de Algoritmos*. Universidad de Málaga.

Karelahti J., 2001. *Solving the cutting stock problem in the steel industry*. Tesis Doctoral, Helsinki University of Technology.

Khalifa Y., Salem O. y Shahin A., 2006. Cutting Stock Waste Reduction Using Genetic Algorithmsm. *GECCO '06*.

Kumar V., 1994. *Introduction to parallel computing*. The Benjamin/Cummings Publishing Company.

Levine J. y Ducatelle F., 2006. Ant Colony Optimization and Local Search for Bin Packing and Cutting Stock Problems. *Université of Edinburg*.

Liberti L., 2008. *Introduction to Global Optimization*. École Polytechnique.

Onwubolu G. y Mutingi M., 2003. A genetic algorithm approach for the cutting stock problem. *Journal of Intelligent Manufacturing*.

Sait S. y Youssef H., 1999. *Iterative Computer Algorithms with Applications in Engineering*. IEEE Computer Society.

Shamblin J. y Stevens G., 1975. *Investigación de Operaciones, un enfoque fundamental*. McGraw Hill.

Sterling T., Becker D., Savarese D., Dorband J., Ranawake U. y Packer C., 1995. BEOWULF: A parallel workstation for scientific computation. En *In Proc. 24th International Conference on Parallel Processing*, págs. 11–14.

Tesharima H., Ross P. y Valenzuela M., 2006. A GA-Based Method to Produce Generalized Hyper-heuristics for the 2D-Regular Cutting Stock Problem. *GECCO '06*.

Turing A., 1936. On computable numbers, with an application to the Entscheidungsproblem. *London Mathematical Society*.

Vidal A. y Pérez J., 2004. *Introducción a la programación en MPI*. Universidad Politécnica de Valencia.

Winston W., 2005. *Investigación de Operaciones -Aplicaciones y Algoritmos-*. Thomson.

Yen C., Wong D. y Jang S., 2004. Solution of trim-loss problem by an integrated simulated annealing and ordinal optimization approach. *Journal of Intelligent Manufacturing*, 12:701–709.

Zolfaghari S. y Liang M., 2002. Comparative study of simulated annealing, genetic algorithms and tabu search for solving binary and comprehensive machine-grouping problems. *International Journal of Production Research*, 40(9):2141–2158.

Apéndice A

Tabla de datos

Tabla A.1: Datos de prueba de 120 pedidos.

Pedido	Día	Ancho	Largo	Cantidad	Pedido	Día	Ancho	Largo	Cantidad
1	1	111,6	225,3	500	24	2	44,8	90,5	1000
2	1	101,6	265,3	500	25	2	43,6	159,7	500
3	1	96,6	195,3	1500	26	2	66,1	171,3	500
4	1	62,7	218,5	500	27	2	71,6	165,3	2500
5	1	38,5	124,3	2000	28	2	75,6	199,3	1000
6	1	56,4	97,3	2000	29	2	32,4	107,3	3000
7	1	43,4	140,5	1500	30	2	42,1	149,3	3000
8	1	53,7	145,9	1500	31	2	38,5	124,3	2000
9	1	49,6	141,3	3000	32	2	43,6	185,3	1000
10	1	106,6	245,3	1000	33	2	51,6	161,3	500
11	1	37,3	120,3	5000	34	2	79,6	185,3	500
12	1	34,1	136,3	4000	35	2	32,6	118,3	3000
13	1	48,0	127,3	500	36	2	34,1	83,3	400
14	1	32,0	96,7	1000	37	2	34,0	83,3	1000
15	1	60,1	146,3	500	38	2	49,1	123,7	1000
16	2	61,8	173,3	500	39	2	47,5	104,7	3000
17	2	57,6	176,3	800	40	2	46,6	118,3	1500
18	2	57,6	177,3	500	41	2	38,7	110,9	500
19	2	47,1	152,3	1000	42	2	42,6	146,3	500
20	2	74,6	220,0	4000	43	2	42,6	146,3	2000
21	2	41,9	140,3	3000	44	2	56,0	123,7	500
22	2	40,1	129,3	1000	45	2	81,5	194,5	500
23	2	45,6	100,9	500	46	3	46,9	158,5	4000

Pedido	Día	Ancho	Largo	Cantidad	Pedido	Día	Ancho	Largo	Cantidad
47	3	61,1	135,3	2000	84	4	122,3	40,7	1000
48	3	103,9	88,1	3000	85	4	39,0	120,1	1500
49	3	56,1	228,3	700	86	4	47,6	221,3	1000
50	3	56,1	228,3	700	87	4	38,6	122,9	2000
51	3	56,1	228,3	700	88	4	57,1	125,3	4000
52	3	38,3	122,9	850	89	4	47,7	128,7	1500
53	3	38,3	122,9	850	90	4	47,1	164,3	2000
54	3	102,1	246,3	500	91	4	55,6	128,3	8500
55	3	103,1	211,3	500	92	4	96,6	195,3	2500
56	3	32,0	96,7	1000	93	4	111,6	225,3	1500
57	3	43,6	185,3	1000	94	4	131,6	285,3	800
58	3	42,5	117,1	1000	95	4	101,6	265,3	700
59	3	53,6	145,3	3000	96	4	153,0	244,0	3000
60	3	49,8	141,7	1000	97	5	65,5	189,9	1500
61	3	38,7	110,9	1000	98	5	57,6	177,3	500
62	3	42,6	146,3	2500	99	5	93,1	217,3	625
63	3	38,1	108,3	800	100	5	39,6	136,9	800
64	3	52,6	127,3	800	101	5	55,8	132,5	5000
65	3	61,4	153,3	500	102	5	35,6	117,3	1000
66	3	42,9	110,9	600	103	5	81,5	194,5	900
67	3	43,1	153,7	2000	104	5	90,6	171,3	500
68	3	40,4	98,3	1000	105	5	58,8	134,1	500
69	4	60,2	174,9	1000	106	5	53,8	139,5	2000
70	4	74,6	227,3	3000	107	5	49,6	141,3	10000
71	4	38,5	124,3	2000	108	5	49,6	141,3	10000
72	4	121,6	221,3	1000	109	5	90,7	181,7	500
73	4	85,8	204,5	500	110	5	50,4	181,9	700
74	4	46,6	118,3	3000	111	5	81,6	245,3	500
75	4	50,6	177,3	500	112	5	49,1	123,7	1000
76	4	36,6	98,5	500	113	5	42,0	118,9	15000
77	4	53,4	123,3	500	114	5	56,9	176,5	1400
78	4	72,6	213,3	500	115	5	53,7	145,9	15000
79	4	93,8	83,5	2000	116	5	56,9	176,5	500
80	4	43,1	125,3	10000	117	5	109,5	222,0	500
81	4	59,0	186,3	1500	118	5	63,1	122,1	1000
82	4	46,9	158,5	4000	119	5	53,6	137,3	700
83	4	73,6	189,3	1000	120	5	37,3	120,3	5000

Tabla A.2: Resultados de las pruebas para el método LP1.

Prueba	Programación	Desperdicio	Patrones	Completados
20.1.	93,77%	3,44%	79	100%
20.2.	93,61%	1,95%	97	100%
20.3.	95,53%	2,31%	64	100%
20.4.	96,29%	3,57%	90	100%
20.5.	94,66%	8,93%	115	100%
20.6.	96,81%	4,28%	106	100%
Promedio 20	95,11%	4,08%	92	100%
40.1.	96,04%	0,51%	96	100%
40.2.	97,38%	1,51%	132	100%
40.3.	96,63%	3,36%	342	100%
40.4.	96,04%	0,88%	170	100%
40.5.	96,29%	0,80%	98	100%
Promedio 40	96,48%	1,41%	168	100%
80.1.	94,38%	0,64%	199	100%
80.2.	98,38%	0,73%	884	100%
80.3.	97,52%	0,65%	538	100%
80.4.	95,86%	0,86%	268	100%
80.5.	95,41%	0,64%	331	100%
Promedio 80	96,31%	0,70%	444	100%
Día 1	95,98%	3,63%	30	100%
Día 2	94,31%	1,01%	87	100%
Día 3	94,00%	2,47%	79	100%
Día 4	95,80%	1,76%	95	100%
Día 5	96,88%	3,77%	165	100%
Total días	95,39%	2,53%	456	100%

Tabla A.3: Resultados de las pruebas para el método LP2.

Prueba	Programación	Desperdicio	Patrones	Completados
20.1.	88,29%	1,57%	21	65%
20.2.	90,40%	1,70%	26	80%
20.3.	83,43%	1,25%	27	75%
20.4.	63,27%	1,37%	31	70%
20.5.	68,84%	1,30%	13	65%
20.6.	44,41%	0,94%	18	55%
Promedio 20	73,11%	1,36%	23	68%
40.1.	110,00%	1,53%	88	100%
40.2.	83,41%	1,23%	102	58%
40.3.	87,00%	1,05%	72	73%
40.4.	105,80%	1,60%	82	98%
40.5.	110,00%	1,83%	87	100%
Promedio 40	99,24%	1,45%	86	86%
80.1.	107,99%	2,12%	214	99%
80.2.	105,94%	1,01%	319	96%
80.3.	109,32%	1,07%	251	100%
80.4.	106,49%	1,19%	232	95%
80.5.	109,87%	1,62%	244	100%
Promedio 80	107,92%	1,40%	252	98%
Día 1	98,03%	1,66%	16	73%
Día 2	109,32%	1,36%	61	97%
Día 3	85,82%	1,47%	33	75%
Día 4	88,17%	1,13%	43	79%
Día 5	59,74%	1,12%	48	52%
Total días	88,21%	1,35%	201	88%

Tabla A.4: Resultados de las pruebas para el método AG1.

Prueba	Programación	Desperdicio	Patrones	Completados
20.1.	102,29%	6,33%	20	75%
20.2.	115,05%	10,84%	28	90%
20.3.	89,54%	6,34%	19	85%
20.4.	59,96%	1,77%	22	70%
20.5.	68,25%	1,06%	18	70%
20.6.	87,11%	14,36%	12	65%
Promedio 20	87,03%	6,78%	20	76%
40.1.	111,17%	3,32%	69	95%
40.2.	97,99%	6,20%	58	80%
40.3.	112,24%	7,49%	59	80%
40.4.	104,16%	3,65%	64	95%
40.5.	110,68%	3,46%	62	98%
Promedio 40	107,25%	4,82%	62	90%
80.1.	105,51%	2,37%	171	94%
80.2.	109,47%	5,05%	190	81%
80.3.	109,91%	5,14%	200	86%
80.4.	104,43%	4,29%	176	81%
80.5.	109,52%	4,73%	178	83%
Promedio 80	107,77%	4,32%	183	85%
Día 1	96,10%	1,61%	15	67%
Día 2	108,86%	3,99%	53	94%
Día 3	81,02%	1,44%	23	64%
Día 4	103,05%	7,93%	46	78%
Día 5	124,33%	20,19%	29	91%
Total días	102,67%	7,03%	166	98%

Tabla A.5: Resultados de las pruebas para el método AG2.

Prueba	Programación	Desperdicio	Patrones	Completados
20.1.	82,50%	1,46%	16	55%
20.2.	82,92%	1,54%	20	65%
20.3.	77,33%	1,13%	21	50%
20.4.	55,73%	1,35%	18	55%
20.5.	65,88%	1,27%	16	60%
20.6.	42,79%	0,95%	13	45%
Promedio 20	67,86%	1,28%	17	55%
40.1.	91,81%	1,32%	64	58%
40.2.	68,49%	1,20%	45	50%
40.3.	81,64%	1,38%	40	55%
40.4.	94,82%	1,51%	63	78%
40.5.	93,41%	1,48%	56	60%
Promedio 40	86,03%	1,38%	54	60%
80.1.	95,32%	1,44%	156	75%
80.2.	88,17%	1,90%	166	63%
80.3.	85,60%	1,44%	162	50%
80.4.	86,09%	1,55%	147	48%
80.5.	81,78%	1,57%	171	50%
Promedio 80	87,39%	1,58%	160	57%
Día 1	90,68%	1,86%	13	67%
Día 2	94,72%	1,42%	42	83%
Día 3	84,94%	1,54%	31	59%
Día 4	79,79%	1,26%	44	63%
Día 5	55,83%	1,07%	30	36%
Total días	81,19%	1,43%	160	79%

Tabla A.6: Resultados de las pruebas para el método híbrido.

Prueba	Programación	Desperdicio	Patrones	Completados
20.1.	100,83%	2,77%	22	85%
20.2.	107,84%	2,56%	28	95%
20.3.	77,00%	1,14%	14	70%
20.4.	64,01%	1,52%	21	70%
20.5.	74,05%	2,22%	21	80%
20.6.	87,43%	2,65%	0,25	85%
Promedio 20	85,19%	2,14%	18	81%
40.1.	103,80%	2,32%	63	93%
40.2.	103,95%	2,17%	58	95%
40.3.	90,25%	2,62%	55	75%
40.4.	102,79%	2,85%	64	93%
40.5.	107,31%	2,29%	62	100%
Promedio 40	101,62%	2,45%	60	91%
80.1.	95,62%	0,99%	116	80%
80.2.	100,19%	1,71%	143	89%
80.3.	102,76%	1,45%	145	93%
80.4.	93,35%	1,33%	107	74%
80.5.	102,58%	1,79%	1,42	95%
Promedio 80	98,90%	1,45%	102	86%
Día 1	96,90%	1,63%	11	73%
Día 2	105,33%	1,76%	41	97%
Día 3	93,72%	2,15%	23	75%
Día 4	93,50%	1,82%	45	88%
Día 5	68,61%	1,67%	25	64%
Total días	91,61%	1,81%	145	92%

Apéndice B

Artículo primero

J. Jaramillo, F. Correa and R. Jaramillo, (2008). *Programación Lineal y Algoritmos Genéticos Para la Solución del Problema de Corte Unidimensional.* Enviado a revisión a la Revista Colombiana de Computación.

Apéndice C

Artículo segundo

J. Jaramillo, F. Correa and R. Jaramillo, (2008). *Método Híbrido Basado en Programación Lineal y Algoritmos Genéticos Para la Solución de un Problema de Corte.* A ser enviado a revisión a la Revista de Ingenierías de la Universidad Eafit.

Índice de Materias