UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

PROGRAMA DE MAESTRÍA Y DOCTORADO EN INGENIERÍA

MAESTRÍA EN INGENIERÍA DE SISTEMAS - INVESTIGACIÓN DE OPERACIONES

TAXONOMÍA Y MÉTODOS DE SOLUCIÓN PARA EL PROBLEMA DE LA MOCHILA

TESIS

QUE PARA OPTAR POR EL GRADO DE:

MAESTRO EN INGENIERÍA

PRESENTA:

ANA LILIA ANAYA MUÑOZ

TUTOR:

DRA. ESTHER SEGURA PÉREZ

FACULTAD DE INGENIERÍA

CIUDAD UNIVERSITARIA, CD. MX. MAYO DE 2023

Dedicatoria ...

A mis papás, Juana Muñoz y Seferino Anaya; por su amor, entrega, perseverancia,... por los principios y valores que siempre me inculcaron, estos han hecho que pueda desempeñarme de la mejor manera en la sociedad, siempre han sido y serán mi ejemplo a seguir, los amo.

A mis hermanas Gaby, Mary y Ara que siempre han sido mi apoyo, las quiero mucho.

A Carmen Hernández Ayuso, desde que fuiste mi profesora en la Facultad de Ciencias despertaste mi interés por la Investigación de Operaciones, has sido un gran apoyo en mi vida.

Agradecimientos

A Dios que me dio la oportunidad de estudiar un posgrado.

A mi familia que siempre han sido mi motor para seguir adelante y alcanzar las metas que me propongo.

A mi tutora Dra. Esther Segura Pérez, gracias a su guía este trabajo pudo llevarse a cabo de la mejor manera.

A mis sinodales Dra. Flores De La Mota Idalia, M.I. Hernandez Ayuso María Del Carmen, Dra. Huerta Barrientos Aida y la Dra. Rodríguez Vázquez Katya; por el tiempo que dedicaron para leer este trabajo de investigación, muchas gracias por sus sugerencias y comentarios que ayudaron a complementarlo.

Y por último pero no menos importante a la Universidad Nacional Autónoma de México que me ha dado las herramientas para desempeñarme en la vida laboral de manera honesta, ética y profesional.

Índice general

Índice de figuras

Índice de cuadros

Introducción

Problema

El problema de la mochila es un problema de programación lineal entera cuyo modelo matemático tiene una estructura sencilla, pero a pesar de esto se considera un problema difícil de resolver.
En este trabajo de investigación se desea desarrollar detalladamente la reducción polinomial partiendo del problema del SAT, hasta llegar al problema de la mochila binario y el general; para demostrar que el problema de la mochila (en su versión de decisión) es NP completo lo cual implica que en su versión de optimización es NP duro.
Cuando un problema de optimización es catalogado como NP-duro, no existen métodos eficientes para resolver todas las instancias de este problema, a partir de esto surge la pregunta ¿Qué método es más eficiente para resolver una instancia del problema de la mochila?, cuestionamiento que sirvió de base para desarrollar este trabajo de investigación
Es una realidad que en México existe una carencia de material escrito (referencias) en español asociadas a los NP-completos y métodos de solución de los problemas enteros como son algoritmos exactos, heurísticos y metaheurísticos; con este trabajo se desea dar un paso para aminorar esta carencia de material

Marco teórico

El concepto de *investigación de operaciones* se remonta a 1938, nació como un término descriptivo para la aplicación de la ciencia a las operaciones militares con el fin de mejorarlas (durante la segunda guerra mundial).
Cuando termina la guerra en 1945 algunos países como La Gran Bretaña utilizaron a la investigación de operaciones para resolver problemas gubernamentales e industriales.
El mayor auge de la investigación de operaciones se da en los años 50*s* en los que se formaron diferentes asociaciones dedicadas únicamente al estudio y desarrollo de ésta, tales como:

- The Operational Research Society (En 1948 en Gran Bretaña)
- The Operational Research Society of America (En 1952 en Estados Unidos)

- La Societe Francaise de Recherche Operationnelle (En 1956 en Francia)
- The international Federation of Operation Research Societies (En 1957)

Algunas definiciones para la investigación de operaciones son:

- Charles Kittel en 1947 (físico estadounidense) la define como: *La investigación de operaciones es un método científico para proporcionar a los departamentos ejecutivos una base cuantitativa para las decisiones*
- En 1962, la definición se había ampliado a: *La investigación de operaciones es el ataque de la ciencia moderna a los problemas complejos que surgen en la dirección y gestión de grandes sistemas de hombres, máquinas, material y dinero en la industria, los negocios, el gobierno y la defensa.*
- Hoy en día *The operational research society* prefiere ilustrar lo que hace la investigación de operaciones mediante ejemplos.

Mientras se desarrollaba la Investigación de Operaciones aparecieron sus ramas, algunas de ellas son:

- Programación entera
- Programación dinámica
- Programación heurística
- Programación lineal
- Programación no lineal
- Simulación
- Teoría de líneas de espera
- Teoría de inventarios
- Teoría de juegos
- Teoría de redes

Nótese la interacción con otros campos de la ciencia dado su origen miltidisciplinario. La programación entera estudia los problemas de optimización matemática cuyas variables asociadas a sus modelos matemáticos son algunas (o todas) enteras. Casi todos los problemas enteros son "difíciles de resolver", es decir, son NP duros; en términos prácticos nos dice que no existen algoritmos que obtengan la solución óptima del problema en un tiempo razonable (Dependiendo del tamaño del problema). Por esta razón han tenido un desarrollo importante los algoritmos heurísticos y metaheurísticos que son rápidos, pero en ocasiones hacen a un lado la eficacia; esto se puede mejorar por medio de cotas y combinación de dos o más algoritmos.
En programación entera no existe un método que resuelva todos los problemas que pertenezcan a esta categoría, tal como existe el método simplex para los problemas

lineales, de aquí la importancia del estudio de métodos o algoritmos que lleguen a una mejor aproximación al óptimo en un tiempo razonable.
Algunos problemas enteros como:

- El problema del agente viajero (TSP)
- El problema de la mochila (KP)
- El problema de localización de plantas
- El problema de transporte
- El problema de empaquetamiento

Se aplican en diversas problemáticas donde se desea optimizar tiempo, rendimiento, distancia, etcétera.
Algunas aplicaciones son: Cargamento, inversiones, transporte, asignación, ruteo de vehículos, producción, corte, etcétera.

El problema de la mochila, también conocido como KP por su abreviatura en inglés Knapsack Problem, llama la atención por la estructura sencilla del modelo matemático; el objetivo del problema es maximizar una función lineal para optimizar el beneficio obtenido por artículos incluidos en un espacio limitado (restricción de capacidad), a pesar de la sencillez de su modelo es catalogado como un problema difícil de resolver (NP-completo); el problema de la mochila es uno de los más estudiados ya que se puede aplicar en varias problemáticas de la vida real, además el modelo matemático puede aparecer como la relajación de un problema cuyo modelo matemático es más complejo; a pesar de que la mayoría de los problemas tienen más de una restricción, en ocasiones es posible darle más importancia a una restricción que a las demás de aquí la ventaja de conocerlo a fondo. Si se explota la estructura sencilla del problema y se aumentan ciertas restricciones que complican el planteamiento, hacen que el modelo sea más cercano a la realidad y esto hace que el estudio del problema de la mochila se vuelva relevante para solucionar problemáticas que involucran la optimización en modelos más complejos, de aquí surgen las variantes del problema de la mochila.

Objetivo

Se demostrará a través de un análisis teórico detallado de los problemas Np Completos que el problema de la mochila pertenece a este conjunto, además del diseño y programación de algunos algoritmos del tipo: constructivo, búsqueda en su entorno, recocido simulado, búsqueda tabú y genético que permitan medir su efectividad al aplicarlo a instancias del problema de la mochila.

Justificación

En México no se le ha dado la suficiente importancia al estudio de este tipo de problemas, uno de los factores puede ser que la mayoría de la bibliografía referente

a los problemas enteros (combinatorios) están en otro idioma (principalmente en inglés), de acuerdo a la literatura revisada, el libro más reciente publicado que es dedicado totalmente al estudio del problema de la mochila es: *Knapsack Problems*, *Hans Kellerer, Ulrich Pferschy, David Pisinger, editorial Springer 2004* , otro factor es que la solución de problemas combinatorios se considera un área relativamente nueva, por lo que las referencias son escasas y esto hace que haya un des interés o un desconocimiento hacia los problemas combinatorios a nivel licenciatura y en ocasiones a nivel posgrado. De aquí la importancia de desarrollar material teórico práctico de diferentes problemas combinatorios que sea accesible y sirva de referencia para estudios de nivel licenciatura y posgrado en idioma español.

Metodología de investigación

A partir de la revisión de diferentes referencias escritas tales como:

- Libros como el *Computers and intractability. A guide to the theory of NP-Completeness, Combinatorial optimization: algorithms and complexity* para la teoría de los NP-completos, *Introducción a la programación lineal* para la teoría de la programación lineal, *Integer Programming*, para la teoría de la programación entera, *Búsqueda y Exploración Estocástica* para la teoría de los algoritmos heurísticos y metaheurísticos, y algunos más; todos contenidos en las referencias.

- Artículos de revistas como *Operations Research, third annual ACM symposium on Theory of computing, Revista Iberoamericana de Inteligencia Artificial, Annals of operations research, Journal of the Operations Research Society of Japan, Springer, Mathematical Programming, INFORMS Journal on Computing* Para los algoritmos heurísticos y metaheurísticos.

- Apuntes de clase de la maestría.

se analizarán a fondo los conceptos y teoría que se desarrollará en este trabajo de investigación, además se diseñarán algunos algoritmos heurísticos y metaheurísticos para el problema de la mochila, éstos se implementarán en instancias del problema de la mochila obtenidas en las páginas:

- `https://people.sc.fsu.edu/~jburkardt/datasets/knapsack_01/knapsack_01.html`

- `http://hjemmesider.diku.dk/~pisinger/codes.html`

se compararán los resultados para así analizar los pros y los contras de cada uno de ellos.
El contenido de este trabajo de investigación contiene 4 capítulos:

- En el capítulo 1 se dará un poco de historia del problema de la mochila, a partir del modelo matemático se analizarán algunas variantes de éste, se desarrollará

a la par un caso aplicado (caso de cargamento) para analizar como cambia el enfoque del modelo matemático al problema.

- En el capítulo 2 se explicarán algunos algoritmos exactos de manera general para después aplicarlos a instancias del problema de la mochila, se analizarán las ventajas y desventajas al aplicarlos.

- En el capítulo 3 se analiza la teoría de los NP-completos, se explicará detalladamente la demostración (la reducción polinomial) partiendo del problema del SAT (primer problema que se demostró que es NP-completo),hasta llegar al problema de la mochila binario y general.

- En el capítulo 4 se explicarán de manera detallada algunos algoritmos heurísticos aplicados al problema de la mochila, también se explicará el diseño de los algoritmos del tipo recocido simulado, búsqueda tabú y genético aplicados también al problema de la mochila; se programarán en el lenguaje Python algunos algoritmos para comparar resultados y así obtener su eficiencia; derivado de éste análisis uno de los objetivos de este trabajo será concluir: cuál de los 4 algoritmos heurísticos y metaheurísticos programados genera una mejor solución al aplicarlo a una instancia de tamaño 24.
- Al final del trabajo se encuentran dos apéndices, el primer apéndice contiene un glosario con conceptos utilizados en el presente trabajo, el segundo contiene los códigos que fueron desarrollados en Python de los programas: Glotón, búsqueda en su entorno, Recocido simulado y Genético.

Capítulo 1

Historia y taxonomía del problema de la mochila

1.1. Historia del problema de la mochila

Los primeros artículos referentes al problema de la mochila se remontan a los años $50's$ que es cuando se desarrolló la Investigación de Operaciones, aunque algunos autores mencionan que en 1897 en el artículo *On the partition of numbers* de *George Mathews Ballard* menciona que un sistema de ecuaciones puede ser reducido, esto puede asociarse al problema de mochila planteado como un problema de decisión.

En este artículo no se aborda el problema como uno de optimización ni tampoco se le nombra como problema de la mochila, hace referencia a la siguiente situación: El problema de la partición de un número multipartito (dividido en varias partes), sean m, m', m", ... números, estos se pueden partir en partes asignadas a, a', a", ...; b, b', b", ...; ..., el problema consiste en encontrar enteros $x, y, z, ...t$, positivos o cero tal que:

$$\begin{gathered} u = ax + by + cz + ... + lt = m \\ u' = a'x + b'y + c'z + ... + l't = m' \\ u'' = a''x + b''y + c''z + ... + l''t = m'' \\ \vdots \end{gathered}$$

En el artículo muestra que es posible reducir las ecuaciones expresándolas como combinación lineal tal que el solucionar

$$(\lambda a + \mu a')x + (\lambda b + \mu b')y + (\lambda c + \mu c')z + (\lambda d + \mu d')w = \lambda m + \mu m' \tag{1.1}$$

(para el caso de 2 ecuaciones) es lo mismo que resolver el sistema:

$$\begin{gathered} u = ax + by + cz + dw = m \\ u' = a'x + b'y + c'z + d'w = m' \end{gathered}$$

para λ y $\mu \in Z^+$

A la ecuación 1.1 se puede asociar al problema de decisión del problema de la mochila el cual dice: Dados los enteros c_j para $j = 1, 2, ..., n$ y k ¿Existen enteros $y_j \geq 0$ para $j = 1, 2, ..., n$ tal que

$$\Sigma_{i=1}^{n} a_i y_i = k?$$

En 1957 *George Dantzing* en el artículo *Discrete-Variable Extremum Problems [Dantzing (1957)]*, hace referencia a ciertos problemas de optimización que él menciona como problemas extremos con variables discretas y aquí incluye al problema de la mochila como uno de ellos, menciona que sí se resuelve como uno de programación lineal (cuyas variables son continuas) en muchos casos el punto extremo óptimo es no entero y se pregunta ¿ Qué pasa si se redondea la solución?

Menciona el procedimiento de Richard Bellman (Programación dinámica) como un método que nos lleva a la solución exacta del problema de la mochila.

En este artículo se refiere al problema de la mochila como el problema que se genera al tratar de resolver la situación común de empacar artículos que denominas más valiosos o útiles sin sobrecargar su equipaje.

En la siguiente sección se planteará el problema de la mochila como un modelo matemático.

1.2. Modelo matemático del problema de la mochila

El nombre de Problema de la Mochila surgió planteando la siguiente situación:

Una persona desea irse de excursión, dispone de una mochila con capacidad b kilogramos y cuenta con diferentes artículos para llenarla; a cada artículo i se le asocia un peso a_i y un beneficio por llevarlo c_i :

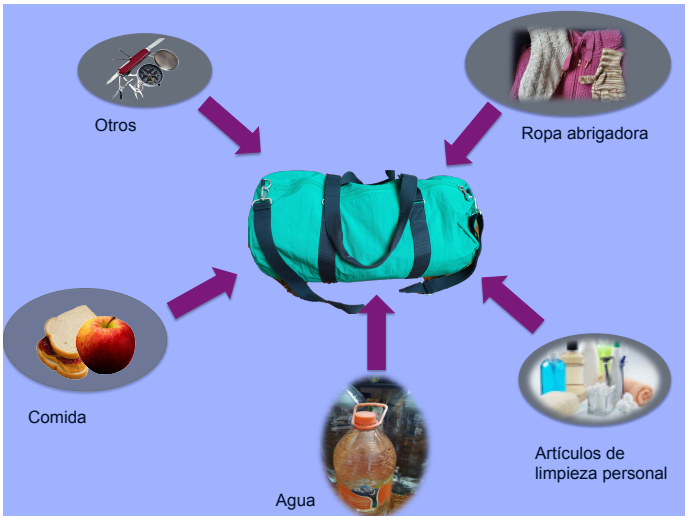


Figura 1.1: *Situación del problema de la mochila*

Fuente: Elaboración propia utilizando Power Point (2011)

A partir de esta problemática surgen dos variantes:

- Si las variables de decisión representan si llevamos o no el artículo, a este problema se le conoce como problema de la mochila binario o Knapsack problem $\{0,1\}$ debido a que las variables son binarias, es decir:

$$x_i = \begin{cases} 1 \quad si \quad se\ incluye\ el\ artículo\ i \\ \\ 0 \quad si \quad no\ se\ incluye\ el\ artículo\ i \end{cases}$$

- Si las variables de decisión representan el número de artículos del tipo i a incluir, a este problema se le conoce como problema de la mochila general ya que las restricciones de las variables son $\forall x_i \in Z$

Tomando en cuenta los siguientes datos:

- b capacidad de la mochila.
- a_i peso unitario por incluir el artículo i.
- c_i beneficio unitario por incluir el artículo i.

el modelo matemático es:

1.2.1. Problema de la mochila binario

Si

$$x_i = \begin{cases} 1 \quad si \quad se\ incluye\ el\ artículo\ i \\ \\ 0 \quad si \quad no\ se\ incluye\ el\ artículo\ i \end{cases}$$

el modelo matemático es:

$$\begin{gathered} max\ z = \Sigma_{i=1}^{n} c_i x_i \\ sujeto\ a \\ \Sigma_{i=1}^{n} a_i x_i \leq b \\ x_i \in \{0,1\} \text{ para } i = 1, ..., n \end{gathered}$$

1.2.2. Problema de la mochila general

Si x_i = Número de artículos del tipo i a incluir
el modelo matemático es:

$$\begin{gathered} max\ z = \Sigma_{i=1}^{n} c_i x_i \\ sujeto\ a \\ \Sigma_{i=1}^{n} a_i x_i \leq b \\ x_i \geq 0\ x_i \in Z \text{ para } i = 1, ..., n \end{gathered}$$

Previo al desarrollo de la taxonomía del problema es necesario conocer algunos conceptos de programación lineal entera, para analizar el comportamiento de las soluciones de un problema entero.

1.3. Definiciones de programación entera

Las soluciones de los problemas de programación lineal entera (PLE) no se comportan igual que las soluciones de los problemas de programación lineal (PL), la principal diferencia radica en la región factible de éstos ya que las variables de un problema de PLE son discretas y las variables de un problema de PL son continuas; esta pequeña diferencia hace que la mayoría de los problemas de PLE sean difíciles de resolver ya que los principales métodos y resultados para resolver un problema de PL no se pueden utilizar para resolver los problemas de PLE.

A pesar de que la diferencia entre los problemas es grande, es útil conocer las bases de la PL para tener herramientas para empezar a analizar los problemas enteros ya que en muchos métodos de solución de PLE se utiliza la PL como primer paso.

1.3.1. Principales definiciones

El modelo matemático de un PPL fue propuesto por George Dantzig en 1947, éste tiene la siguiente estructura:

$$
\begin{gathered}
max\ z = \Sigma_{i=1}^{n} c_i x_i \\
sujeto\ a \\
\Sigma_{i=1}^{n} a_{1i} x_i \leq b_1 \\
\Sigma_{i=1}^{n} a_{2i} x_i \leq b_2 \\
\vdots \\
\Sigma_{i=1}^{n} a_{ji} x_i \leq b_m \\
x_i \geq 0 \text{ para } i = 1, \ldots, n
\end{gathered}
$$

El modelo matemático de un problema de PLE es similar al modelo anterior solo se le agrega la restricción a las variables $x_i \in Z\ \forall i$ (o para algunas) o $x_i \in \{0, 1\}$ para $i = 1, ..., n$

Problema entero (general)

$$
\begin{gathered}
max\ z = \Sigma_{i=1}^{n} c_i x_i \\
sujeto\ a \\
\Sigma_{i=1}^{n} a_{1i} x_i \leq b_1 \\
\Sigma_{i=1}^{n} a_{2i} x_i \leq b_2 \\
\vdots \\
\Sigma_{i=1}^{n} a_{ji} x_i \leq b_m \\
\forall x_i \geq 0 \\
x_i \in Z \text{ para } i = 1, ..., n
\end{gathered}
$$

Problema entero (binario)

$$
\begin{gathered}
max\ z = \Sigma_{i=1}^{n} c_i x_i \\
sujeto\ a \\
\Sigma_{i=1}^{n} a_{1i} x_i \leq b_1 \\
\Sigma_{i=1}^{n} a_{2i} x_i \leq b_2 \\
\vdots \\
\Sigma_{i=1}^{n} a_{ji} x_i \leq b_m \\
\forall x_i \geq 0 \\
x_i \in \{0, 1\} \text{ para } i = 1, ..., n
\end{gathered}
$$

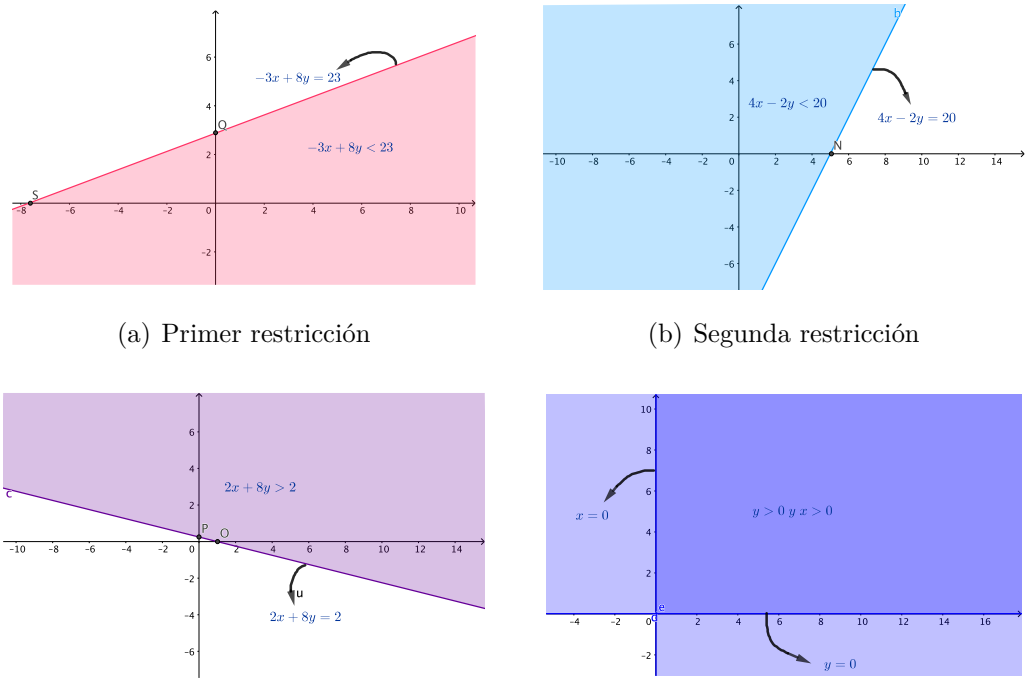
Ahora hablemos de la región factible de un problema de PLE, para esto primero definiremos lo que es una solución factible

Definición 1 *Una **solución factible** de un problema de PL o PLE es un punto $X \in R^n$ cuyas entradas satisfacen todas las restricciones del problema, incluyendo*

las restricciones de las variables, al conjunto de todos puntos factibles se le conoce como ***región factible*** *y se denota como* F_P.

Ejemplo 1 *Para ilustrar este concepto, consideremos el siguiente problema de PL:*

$$
\begin{gathered}
max\ z = 4x + 9y \\
sujeto\ a \\
-3x + 8y \leq 23 \\
4x - 2y \leq 20 \\
2x + 8y \geq 2 \\
x, y \geq 0
\end{gathered}
$$



(a) Primer restricción

(b) Segunda restricción

(c) Tercer restricción

(d) No negatividad

Figura 1.2: *Puntos que satisfacen las restricciones del problema de PL*

Fuente: Elaboración propia utilizando Geogebra 5 (2016)

La región factible del problema es la intersección de los subespacios anteriores.

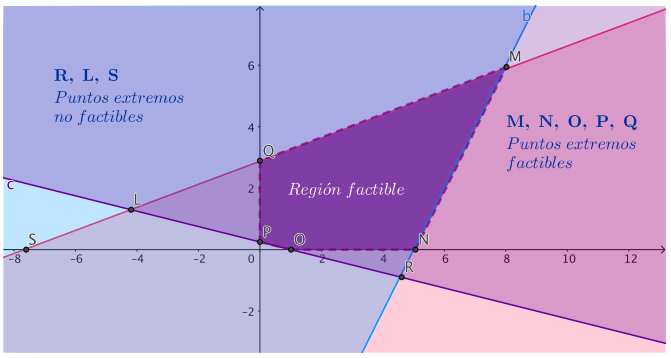


Figura 1.3: *Región factible del problema de PL*

Fuente: Elaboración propia utilizando Geogebra 5 (2016)

Observación 1 *La región factible de un problema de PL es un conjunto convexo, la demostración se puede encontrar en el libro de [Hernández (*2013*)].*

Definición 2 *X^* es la solución óptima de un problema de PL si es una solución factible que optimiza la función objetivo z.*

Esto nos da pie al siguiente resultado.

Teorema 1 *La solución óptima, si existe, la alcanza en un* ***punto extremo*** *de la región factible, es decir en uno de los vértices del conjunto convexo que forma la región factible, la demostración se puede encontrar en el libro de [Hernández (*2013*)].*

El óptimo lo alcanza en el punto:

$$X^* = (7.92, 5.84)$$

y el valor de la función objetivo al evaluar X^* es:

$$z(X^*) = 84.3076$$

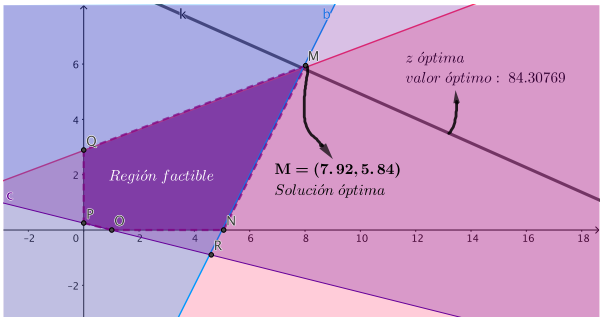


Figura 1.4: *X y z óptimas del problema de PL*

Fuente: Elaboración propia utilizando Geogebra 5 (2016)

Ahora definimos algunos conceptos para un problema de PLE.

La **región factible de un problema de PLE** es un conjunto discreto numerable. Para ilustrar esto, consideremos el siguiente ejemplo.

Ejemplo 2 *Si consideramos el problema anterior, pero incluimos la restricción de que las variables deben ser enteras, es decir:*

$$
\begin{gathered}
max\ z = 4x + 9y \\
sujeto\ a \\
-3x + 8y \leq 23 \\
4x - 2y \leq 20 \\
2x + 8y \geq 2 \\
x, y \geq 0 \\
x, y \in Z
\end{gathered}
$$

La región factible es:

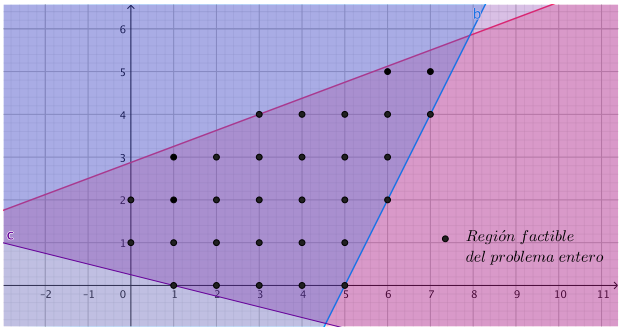


Figura 1.5: *Región factible del problema entero*

Fuente: Elaboración propia utilizando Geogebra 5 (2016)

Definición 3 ***La relajación PL de un problema de PLE*** *es el mismo modelo matemático pero omitiendo las restricciones de que las variables deben ser enteras, quedando así un PPL.*

Proposición 1 *La región factible de un problema de PLE tiene la propiedad de estar contenida en la región factible de la relajación PL del problema entero.*

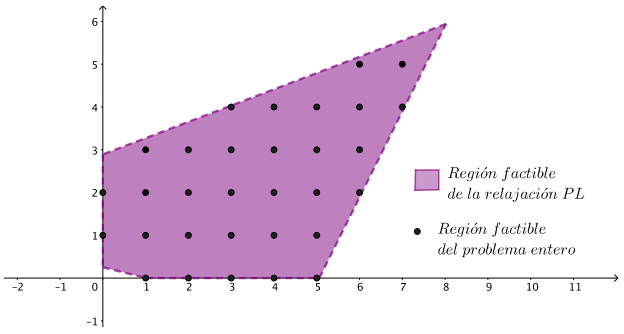


Figura 1.6: *Contención de la región factible del problema entero en la región factible del problema de PL*

Fuente: Elaboración propia utilizando Geogebra 5 (2016)

Observación 2 *La propiedad de contención entre las regiones factibles es importante ya que proporciona una cota superior sobre la función objetivo z (en el caso de que el problema es a maximizar) o una cota inferior sobre la función objetivo z (en el caso de que la función objetivo sea a minimizar)*

Ejemplo 3 *Las soluciones óptimas de los respectivos problemas son:*

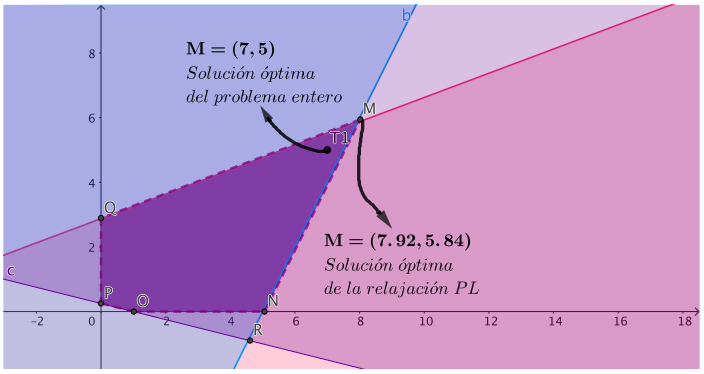


Figura 1.7: *Soluciones óptimas del problema entero y el problema de PL*

Fuente: Elaboración propia utilizando Geogebra 5 (2016)

Donde el valor de la función objetivo de la relajación PL es :

$$z(7.92, 5.84) = 84.30769$$

La función objetivo del problema entero es:

$$z(7, 5) = 73$$

Al comparar ambos resultados:

$$z(7.92, 5.84) = 84.30769 > 73 = z(7, 5)$$

Lo cual cumple con la propiedad de ser una cota superior de la función objetivo.

Si la solución de la relajación PL proporciona una cota superior, nos hace pensar... ¿y si redondeamos?, podemos caer en 2 casos:

- Si redondeamos hacia el siguiente entero, $X = (8, 6)$ no sería una solución factible, ya que no cumple con la primer restricción:

$$-3x + 8y \leq 23$$

$$-3(8) + 8(6) = 24 > 23$$

- Si redondeamos al entero menor, $X = (7, 5)$ en este caso coincidió con la solución del problema entero, pero son pocos los casos en los que ocurre esto.

Observación 3 *El redondeo no es un método eficaz para obtener el óptimo de un problema de PLE, en algunos casos caemos en un punto fuera de la región factible, en otros casos no caemos en el óptimo (aunque si está en la región factible) y en muy pocos casos caemos en el óptimo.*

Definición 4 *Una instancia de un problema de optimización es una pareja* (F, c)*, donde:*

- F *es el conjunto de soluciones factibles.*
- c *es la función de costo.* $c : F \to R^1$ *función que se desea optimizar.*

Obtener el óptimo de una instancia de un problema es encontrar $f \in F$ tal que $c(f) \geq c(y)\ \forall y \in F$
f es llamada solución óptima del problema (Para un problema de maximización).

Ejemplo 4 *Los siguientes problemas son instancias del problema de la mochila:*

- $n = 5;\ c = \{24, 13, 23, 15, 16\};\ a = \{12, 7, 11, 8, 9\};\ b = 26$
- $n = 7;\ c = \{70, 20, 39, 37, 7, 5, 10\};\ a = \{31, 10, 20, 19, 4, 3, 6\};\ b = 50$

Cuyos modelos matemáticos son:

- $max\ z = 24x_1 + 13x_2 + 23x_3 + 15x_4 + 16x_5$
  $sujeto\ a$
  $12x_1 + 7x_2 + 11x_3, +8x_4 + 9x_5 \leq 26$
  $x_i \in \{0, 1\}\ para\ i = 1, 2, 3, 4, 5$

- $max\ z = 70x_1 + 20x_2 + 39x_3 + 37x_4 + 7x_5 + 5x_6 + 10x_7$
  $sujeto\ a$
  $31x_1 + 10x_2 + 20x_3, +19x_4 + 4x_5 + 3x_6 + 6x_7 \leq 50$
  $x_i \in \{0, 1\}\ para\ i = 1, 2, ..., 7$

respectivamente.

Definición 5 *Un problema de optimización se compone del conjunto de todas las instancias del problema.*

Definición 6 *Un problema es combinatorio si el conjunto F está formado por soluciones discretas.*

Observación 4 *El problema de la mochila es un problema de optimización combinatoria*

Observación 5 *El optimizar un problema combinatorio nos hace pensar que el óptimo se podría obtener de manera sencilla, ya que las soluciones factibles están en un conjunto numerable y bastaría con probar todas esas soluciones y la mejor será la óptima; este procedimiento resulta sencillo y relativamente rápido, pero el conjunto de soluciones puede crecer de manera exponencial en instancias grandes y por esto el procedimiento para obtener el óptimo crece de la misma manera y los convierte en problemas intratables, es decir, que nos tardaríamos mucho tiempo para poder encontrar el óptimo.*

La complejidad del problema de la mochila radica en que si aumenta el número de artículos en una instancia, el número de alternativas para la solución del problema crece de manera exponencial; por ejemplo, si el problema es binario el número de alternativas para cada instancia es:

- Si una instancia tiene 5 artículos, el número de alternativas a evaluar en la función objetivo son: $2^5 = 32$
- Si una instancia tiene 8 artículos, el número de alternativas a evaluar en la función objetivo son: $2^8 = 256$
- Si una instancia tiene 10 artículos, el número de alternativas a evaluar en la función objetivo son: $2^{10} = 1024$
- Si una instancia tiene 20 artículos, el número de alternativas a evaluar en la función objetivo son: $2^{20} = 1,048,576$
- Si una instancia tiene 24 artículos, el número de alternativas a evaluar en la función objetivo son: $2^{24} = 16,777,216$

¿Cuánto nos tardaríamos en evaluar $16,777,216$ alternativas en la función objetivo z y después compararlas para escoger la mejor?

Por esta razón los problemas combinatorios como el del agente viajero y el problema de la mochila, resultan intratables cuando se trata de instancias grandes y los métodos exactos resultan ser poco eficientes (con respecto al tiempo) y de aquí la importancia del desarrollo de algoritmos rápidos como los heurísticos y metaheurísticos que con el paso del tiempo se han modificado para proporcionar una solución cercana al óptimo.

1.4. Taxonomía del problema de la mochila

En esta sección se explicarán algunas de las variantes del problema de la mochila, además se llevará a la par un ejemplo aplicado, explicando cómo se modifica el problema cuando pasa de una variante a otra.

Tomemos en cuenta el siguiente problema: Supongamos que Juan tiene una tienda que también es verdulería, es decir, en ella se vende frutas y verduras; él dispone de una camioneta *RAM 700 cabina sencilla* la cual tiene una capacidad de carga 705 kilogramos, a Juan no le gusta llegar a su capacidad máxima de carga y siempre deja 40 kg libres, entonces, la camioneta reduce su capacidad a 665 kg, cada semana va a surtirse de frutas y verduras, tiene una lista de 49 diferentes artículos diferentes, cada uno de ellos está empacado en diferentes presentaciones, puede ser por caja, por costal, por manojo o por bolsa, existen 28 artículos que son indispensables, éstos se muestran en la siguiente tabla:

Artículos que son indispensables

Cuadro 1.1: *Artículos y peso por empaque (1)*

Artículo	**Peso**
Aguacate Hass	10 Kg
Ajo Morado	3kg
Apio	8 Kg
Calabacita Ital.	25 Kg
Cebolla Bola	30 Kg.
Chayote Sin Esp	18 Kg
Chicharo	5kg
Chile Poblano	5kg
Chile Serrano	3kg
Cilantro	5 Kg
Ejote Redondo	5KG
Elote Grande	2 docenas /10kg
Epazote	5 Kg
Espinaca	5 Kg

Cuadro 1.2: *Artículos y peso por empaque (2)*

Artículo	**Peso**
Lechuga Romana	docena/8kg
Limón C/semilla	19 Kg
Manzana Gold. Del.	20 Kg
Manzana Red Del.	20 Kg
Naranja Valen. Med.	35 Kg.
Nopal	ciento/10kg
Papa Alpha	50 Kg
Papaya Maradol	17 Kg
Pepino	25 kg
Pera D'anjou	18 Kg
Plátano Tabasco	18 Kg
Tomate Saladette	3 cajas 39 kg
Tomate Verde	28 Kg
Zanahoria Med	28kg

Como son artículos indispensables, no pueden faltar, por lo tanto la suma de los pesos se restará de la capacidad de carga, quedando lo siguiente:

$$665\ kg - 472\ kg = 193\ kg$$

El resto de la capacidad debe ser ocupada con los 19 artículos restantes, a estos artículos se les asignará un beneficio el cual está en función de la demanda del artículo:

Cuadro 1.3: *Peso y beneficio de artículos a incluir en el cargamento*

Artículo	**Peso**	**Beneficio**
1. Ciruela Roja	13 Kg	5
2. Durazno Amarillo	13 Kg	7
3. Fresa	5 Kg	8
4. Guayaba	13 Kg	9
5. Jicama Mediana	20 Kg	9
6. Limón Sin Semilla	20 Kg	5
7. Melón Chino	17 Kg	9
8. Acelga	5 Kg	10
9. Cebolla de Rabo	5 Kg.	7
10. Piña Mediana	17 Kg	9
11. Plátano Dominico	17 Kg	8
12. Plátano Macho	17 Kg	6
13. Sandía Sangria	18 Kg	5
14. Toronja Roja	35 Kg.	9
15. Uva Globo	8 Kg	7
16. Uva Thompson	8 Kg	9
17. Tomate Bola	10 kg	10
18. Chile Jalapeño	3kg	7
19. Chile de Árbol	3kg	6

Al sumar los pesos de todos los artículos obtenemos: 247 kg que es mayor a la capacidad disponible 193 kg por lo que se debe elegir qué artículos incluir.

Tomando en cuenta este contexto se plantearán las variantes del problema de la mochila, utilizando la información anterior:

1.5. Problema de la mochila binario

Este modelo matemático utiliza la siguiente definición de las variables de decisión:

$$x_i = \begin{cases} 1 \quad si \quad \textit{se incluye el artículo } i \\ \\ 0 \quad si \quad \textit{no se incluye el artículo } i \end{cases}$$

el modelo matemático es:

$$max\ z = \Sigma_{i=1}^{n} c_i x_i$$
$$sujeto\ a$$
$$\Sigma_{i=1}^{n} a_i x_i \leq b$$
$$x_i \in \{0,1\} \text{ para } i = 1, ..., n$$

1.5.1. Aplicación

En esta variante la variable de decisión solo nos indica si se compra o no el artículo i, estas se definen:

$$x_i = \begin{cases} 1 & si \quad se\ compra\ la\ verdura\ o\ fruta\ i \\ 0 & si \quad no\ se\ compra\ la\ verdura\ o\ fruta\ i \end{cases}$$

para $i = 1, \ldots, 19$

$$max\ z = 5x_1 + 7x_2 + 8x_3 + 9x_4 + 9x_5 + 5x_6 + 9x_7 + 10x_8 + 7x_9 + 9x_{10} + 8x_{11} + 6x_{12} + 5x_{13} + 9x_{14} + 7x_{15} + 9x_{16} + 10x_{17} + 7x_{18} + 6x_{19}$$
$$sujeto\ a$$
$$13x_1 + 13x_2 + 5x_3 + 13x_4 + 20x_5 + 20x_6 + 17x_7 + 5x_8 + 5x_9 + 17x_{10} + 17x_{11} + 17x_{12} + 18x_{13} + 35x_{14} + 8x_{15} + 8x_{16} + 10x_{17} + 3x_{18} + 3x_{19} \leq 193$$
$$x_i \in \{0, 1\} \text{ para } i = 1, \ldots, 19$$

1.6. Problema de la mochila general

Este problema define a las variables de decisión de la siguiente manera:

$$x_i = \#\ de\ artículos\ del\ tipo\ i\ a\ incluir\ en\ la\ mochila$$

Por lo que el modelo matemático es:

$$max\ z = \Sigma_{i=1}^{n} c_i x_i$$
$$sujeto\ a$$
$$\Sigma_{i=1}^{n} a_i x_i \leq b$$
$$x_i \geq 0 \text{ y } x_i \in Z \text{ para } i = 1, ..., n$$

1.6.1. Aplicación

En este problema, las variables de decisión pueden tomar cualquier valor entero positivo, es decir, nos indican cuantos artículos del tipos i se van a incluir, estas se definen:

$$x_i = \#\ de\ cajas,\ costales,\ manojos\ o\ bolsas\ a\ comprar\ del\ artículo\ i$$

$$max\ z = 5x_1 + 7x_2 + 8x_3 + 9x_4 + 9x_5 + 5x_6 + 9x_7 + 10x_8 + 7x_9 + 9x_{10} + 8x_{11} + 6x_{12} + 5x_{13} + 9x_{14} + 7x_{15} + 9x_{16} + 10x_{17} + 7x_{18} + 6x_{19}$$
$$sujeto\ a$$
$$13x_1 + 13x_2 + 5x_3 + 13x_4 + 20x_5 + 20x_6 + 17x_7 + 5x_8 + 5x_9 + 17x_{10} + 17x_{11} + 17x_{12} + 18x_{13} + 35x_{14} + 8x_{15} + 8x_{16} + 10x_{17} + 3x_{18} + 3x_{19} \leq 193$$
$$x_i \geq 0 \text{ y } x_i \in Z \text{ para } i = 1, \ldots, 19$$

A partir de estos, se pueden generar diferentes planteamientos de problemas tipo mochila, esto es, aumentando algunas restricciones que en algunos casos suelen complicar el modelo pero hace que estos sean más cercanos a la realidad. A continuación se explicarán algunas de las variantes del problema de la mochila.

1.7. Problema subset-sum

Es un caso particular del problema de la mochila binario, la diferencia entre ambos es que el peso de cada artículo es igual a su beneficio, es decir, $c_i = a_i \ \forall i$, por lo que el modelo matemático asociado a este problema es:

$$Max\ z = \Sigma_{i=1}^{n} c_i x_i$$
$$sujeto\ a$$
$$\Sigma_{i=1}^{n} c_i x_i \leq b$$
$$x_i = \{0, 1\} \text{ para } i = 1, ..., n$$

1.7.1. Aplicación

En este caso el beneficio de cada artículo es igual a su peso; esto quiere decir que todos los artículos tienen el peso igual a la demanda, es decir, lo único que nos interesa es maximizar el cargamento tomando en cuenta el peso por empaque y la capacidad de la camioneta; el modelo matemático asociado a este problema es:

$$max\ z = 13x_1 + 13x_2 + 5x_3 + 13x_4 + 20x_5 + 20x_6 + 17x_7 + 5x_8 + 5x_9 + 17x_{10} + 17x_{11} + 17x_{12} + 18x_{13} + 35x_{14} + 8x_{15} + 8x_{16} + 10x_{17} + 3x_{18} + 3x_{19}$$
$$sujeto\ a$$
$$13x_1 + 13x_2 + 5x_3 + 13x_4 + 20x_5 + 20x_6 + 17x_7 + 5x_8 + 5x_9 + 17x_{10} + 17x_{11} + 17x_{12} + 18x_{13} + 35x_{14} + 8x_{15} + 8x_{16} + 10x_{17} + 3x_{18} + 3x_{19} \leq 193$$
$$x_i = \{0, 1\} \text{ para } i = 1, \ldots, 19$$

1.8. Problema de la mochila acotado.

Este problema surge de la siguiente situación: Supongamos que del artículo i se puede llevar a lo más d_i unidades.
El modelo matemático es:

$$Maxz = \Sigma_{i=1}^{n} c_i x_i$$
$$sujeto\ a$$
$$\Sigma_{i=1}^{n} a_i x_i \leq b$$
$$x_i \in Z \text{ y } 0 \leq x_i \leq d_i \text{ para } i = 1, ..., n$$

1.8.1. Aplicación

Para este caso supongamos que se tienen las siguientes restricciones:

- Si se compra acelga a lo más deben ser 3 manojos. $x_8 \leq 3$
- Si se compra piña a lo más deben ser 2 cajas. $x_{10} \leq 2$
- Si se compra uva globo a lo más deben ser 3 cajas. $x_{15} \leq 3$
- Si se compra tomate bola a lo más deben ser 2 cajas. $x_{17} \leq 2$

El modelo matemático que representa la problemática es:

$$max\ z = 5x_1 + 7x_2 + 8x_3 + 9x_4 + 9x_5 + 5x_6 + 9x_7 + 10x_8 + 7x_9 + 9x_{10} + 8x_{11} + 6x_{12} + 5x_{13} + 9x_{14} + 7x_{15} + 9x_{16} + 10x_{17} + 7x_{18} + 6x_{19}$$
$$sujeto\ a$$
$$13x_1 + 13x_2 + 5x_3 + 13x_4 + 20x_5 + 20x_6 + 17x_7 + 5x_8 + 5x_9 + 17x_{10} + 17x_{11} + 17x_{12} + 18x_{13} + 35x_{14} + 8x_{15} + 8x_{16} + 10x_{17} + 3x_{18} + 3x_{19} \leq 193$$
$$x_8 \leq 3$$
$$x_{10} \leq 2$$
$$x_{15} \leq 3$$
$$x_{17} \leq 2$$
$$x_i \geq 0 \text{ y } x_i \in Z \text{ para } i = 1, \ldots, 19$$

1.9. Problema Change-making

Es un caso particular del problema de la mochila acotado y surge de la condición $c_i = 1$ para $i = 1, .., n$, es decir, solo se desea maximizar el número de artículos incluidos en la mochila.
El modelo matemático que representa este problema es:

$$Max\ z = \Sigma_{i=1}^{n} x_i$$
$$sujeto\ a.$$
$$\Sigma_{i=1}^{n} a_i x_i \leq b$$
$$x_i \in Z \text{ y } 0 \leq x_i \leq d_i \text{ para } i = 1, ..., n$$

1.9.1. Aplicación

La situación la podemos plantear como en la sección anterior, la diferencia es que ahora no nos importa la demanda de los artículos los trataremos por igual, lo que nos interesa en incluir la máxima cantidad posible de artículos; el modelo matemático es:

$$max\ z = \Sigma_{i=1}^{19} x_i$$
$$sujeto\ a$$
$$13x_1 + 13x_2 + 5x_3 + 13x_4 + 20x_5 + 20x_6 + 17x_7 + 5x_8 + 5x_9 + 17x_{10} + 17x_{11} + 17x_{12} + 18x_{13} + 35x_{14} + 8x_{15} + 8x_{16} + 10x_{17} + 3x_{18} + 3x_{19} \leq 193$$
$$x_8 \leq 3$$
$$x_{10} \leq 2$$
$$x_{15} \leq 3$$
$$x_{17} \leq 2$$
$$x_i \geq 0 \text{ y } x_i \in Z \text{ para } i = 1, \ldots, 19$$

1.10. Multiple choice knapsack problem

Este modelo surge de la situación en que existen m_i diferentes marcas de cada tipo y a lo más uno de ellos es seleccionado, cada marca tiene su propio peso y beneficio (medido por un número real). Nuestro problema es decidir la marca de

cada artículo que se va a incluir para que el beneficio total se maximice tal que se cumpla la restricción de peso (capacidad).

El modelo matemático asociado a este planteamiento es:

$$maxz = \Sigma_{i=1}^{n}\Sigma_{j=1}^{m_i} c_{ij}x_{ij}$$
$$sujeto\ a$$
$$\Sigma_{i=1}^{n}\Sigma_{j=1}^{m_i} a_{ij}x_{ij} \leq b$$
$$x_{ij} = \{0,1\} \text{ para } i = 1,...,n \text{ y } j = 1,2,...,m_i$$

Como máximo uno de los $x_{i1}, x_{i2}, ..., x_{im_i}$ es positivo, para $i = 1, ...n$
Para resolver este problema se requiere seleccionar como máximo un elemento de cada uno de los de n grupos (cada grupo i tiene m_i artículos diferentes).

1.10.1. Aplicación

La situación que ejemplifica esta variante es:

Supongamos tenemos 3 artículos diferentes: Tomate, Fresa y Uva. (Para no manejar tantos datos)
De cada una de ellas se tienen 3 opciones de compra, cada una de ellas se empacan en cajas de diferentes pesos y también tendrán un beneficio diferente ya que los clientes solicitan más una marca que otra.

Cuadro 1.4: *Diferentes presentaciones del tomate bola*

Lugar de procedencia	Peso de empaque	Beneficio
1. Sinaloa	10 kg	10
2. Nayarit	14kg	8
3. Importación	11 kg	9

Cuadro 1.5: *Diferentes presentaciones de la fresa*

Lugar de procedencia	Peso de empaque	Beneficio
1. Michoacan	5 kg	7
2. Guanajuato	7kg	8
3. Guanajuato	9 kg	8

Cuadro 1.6: *Diferentes presentaciones de la uva*

Lugar de procedencia	Peso de empaque	Beneficio
1. Importación	8 kg	6
2. Zacatecas	9kg	7
3. Zacatecas	5 kg	7

La información de los cuadros anteriores se puede resumir en una matriz de pesos y una de beneficios:

Cuadro 1.7: *Matriz de pesos*

	Alt. 1	Alt. 2	Alt. 3
1. Tomate bola	10	14	11
2. Fresa	5	7	9
3. Uva	8	9	5

Cuadro 1.8: *Matriz de beneficios*

	Alt. 1	Alt. 2	Alt. 3
1. Tomate bola	10	8	9
2. Fresa	7	8	8
3. Uva	6	7	7

La definición de las variables de decisión es:

$$x_{ij} = \begin{cases} 1 \quad si \quad \textit{se incluye el artículo } i \textit{ de la alternativa } j \\ \\ 0 \quad si \qquad\qquad\qquad \textit{no se incluye} \end{cases}$$

para $i = 1, 2, 3$ y $j = 1, 2, 3$
Si suponemos que tenemos disponibles 20 kg para estos artículos, el modelo matemático asociado a este problema es:

$$max\ z = 10x_{11} + 8x_{12} + 9x_{13} + 7x_{21} + 8x_{22} + 8x_{23} + 6x_{31} + 7x_{32} + 7x_{33}$$
$$sujeto\ a$$
$$10x_{11} + 14x_{12} + 11x_{13} + 5x_{21} + 7x_{22} + 9x_{23} + 8x_{31} + 9x_{32} + 5x_{33} \leq 20$$
$$x_{11} + x_{12} + x_{13} \leq 1$$
$$x_{21} + x_{22} + x_{23} \leq 1$$
$$x_{31} + x_{32} + x_{33} \leq 1$$
$$x_{ij} = \{0, 1\} \text{ para } i = 1, 2, 3 \text{ y } j = 1, 2, 3$$

Las últimas 3 restricciones nos garantizan que solo se elegirá una sola alternativa por artículo.

Algunos de los primeros artículos que se refieren a este problema son:

- Ibaraki, T., Hasegawa, T., Teranaka, K., & Iwase, J. (1978). The multiple-choice knapsack problem. Journal of the Operations Research Society of Japan, 21(1), 59-95.

- Zemel, E. (1980). The linear multiple choice knapsack problem. Operations Research, 28(6), 1412-1423.

1.11. Problema de la mochila con restricciones disjuntivas

El problema de la mochila con restricciones disjuntivas (DCKP) o también conocido como problema de la mochila con conflicto en gráficas (KCG) es una extensión del problema de mochila binario $(0-1)$ que además de la restricción de capacidad se aumentan restricciones de incompatibilidad entre pares de artículos. Esto significa que a partir de cada uno de estos pares en conflicto a lo más un elemento puede ser empacado en la mochila. Es posible representar estas restricciones por medio de una gráfica (de conflicto) no dirigida $G = [X, A]$, donde cada vértice corresponde a un artículo del problema y la arista $(i, j) \in A$ indica que los artículo i y j no pueden ser empacados juntos.
La gráfica $G = [V, A]$ de conflicto no necesariamente debe ser conexa, es decir, puede contener vértices aislados lo cual indica que los artículos se pueden combinar sin restricción.
El modelo matemático se representa como sigue:

$$Maxz = \Sigma_{i=1}^{n} c_i x_i$$
$$sujeto\ a$$
$$\Sigma_{i=1}^{n} a_i x_i \leq b$$
$$x_i + x_j \leq 1$$
$$x_i = \{0, 1\} \text{ para } i = 1, ..., n$$

Donde los artículos i y j entran en conflicto, es decir, solo se puede empacar uno de ellos.

Observación 6 *El problema KCG también puede verse como una generalización del conjunto independiente (o conjunto estable), este problema surge de una gráfica $G = [X, A]$ donde se pide un conjunto máximo de vértices que no son adyacentes entre sí.*

Observación 7 *El problema KCG es NP-duro y no permite algoritmos de tiempo polinomial.*

Uno de los primeros artículos donde se mencionó el problema de la mochila con restricciones disjuntivas fue en 2002:
Yamada, T., Kataoka, S., & Watanabe, K. (2002). Heuristic and exact algorithms for the disjunctively constrained knapsack problem. Information Processing Society of Japan Journal, 43(9).
Las gráficas de conflicto aplicadas en problemas de optimización combinatoria que están relacionados con el problema de la mochila, son:

- El problema de empaquetamiento binario con conflictos (BPCG). En este caso, las aristas de la gráfica de conflicto definen pares de elementos que no están permitidos en el mismo envase, es decir, el empaquetamiento de cada contenedor debe ser un conjunto independiente (estable) con respecto a la gráfica dada.

- Un planteamiento diferente es considerada en el artículo de [Baker (1996)]. Donde las aristas de la gráfica de conflicto indican pares de trabajos que no se pueden ejecutar en el mismo intervalo de tiempo (pero posiblemente en la misma máquina). Muestran que el problema puede ser resuelto en tiempo polinomial si la gráfica de conflicto es un bosque.

Algunos de los algoritmos utilizados para resolver este problema es:

- Ramificación y acotamiento
- Algoritmo de búsqueda local
- Recocido simulado
- Algoritmo genético

1.11.1. Aplicación

Supongamos que tenemos el planteamiento del problema binario de la primer sección, pero se agregarán algunas restricciones:

- No se pueden comprar al mismo tiempo melón y sandía, es decir:

$$x_7 + x_{13} \leq 1$$

- No se pueden comprar al mismo tiempo chile jalapeño y chile de árbol, es decir:

$$x_{18} + x_{19} \leq 1$$

- No se pueden comprar al mismo tiempo uva globo y uva thompson, es decir:

$$x_{15} + x_{16} \leq 1$$

El modelo matemático es:

$$max\ z = 5x_1 + 7x_2 + 8x_3 + 9x_4 + 9x_5 + 5x_6 + 9x_7 + 10x_8 + 7x_9 + 9x_{10} + 8x_{11} + 6x_{12} + 5x_{13} + 9x_{14} + 7x_{15} + 9x_{16} + 10x_{17} + 7x_{18} + 6x_{19}$$
$$sujeto\ a$$
$$13x_1 + 13x_2 + 5x_3 + 13x_4 + 20x_5 + 20x_6 + 17x_7 + 5x_8 + 5x_9 + 17x_{10} + 17x_{11} + 17x_{12} + 18x_{13} + 35x_{14} + 8x_{15} + 8x_{16} + 10x_{17} + 3x_{18} + 3x_{19} \leq 193$$
$$x_7 + x_{13} \leq 1$$
$$x_{18} + x_{19} \leq 1$$
$$x_{15} + x_{16} \leq 1$$
$$x_i \in \{0, 1\} \text{ para } i = 1, \ldots, 19$$

1.12. Multiple Knapsacks Problema $0/1$(Problema de varias mochilas)

El problema de la mochila tiene una variante multidimensional, esta variante tiene m restricciones de tipo mochila, pero además tiene la restricción *Cada artículo si se coloca, se coloca en todas las mochilas o en ninguna* . Tradicionalmente, el problema de la mochila unidimensional se resuelve mediante la programación dinámica o ramificación y acotamiento, el problema multidimensional suele reducirse a uno unidimensional mediante el uso de multiplicadores lagrange que, generalmente no proporcionan la solución exacta al problema planteado.
El problema de la mochila multidimensional (MKP) es un problema de optimización combinatoria.

El modelo matemático es:

$$\begin{gathered} Maxz = \Sigma_{i=1}^{n} c_i x_i \\ sujeto\ a \\ \Sigma_{i=1}^{n} a_{ji} x_i \leq b_j \text{ para } j = 1, 2, ..., m \\ x_i = \{0, 1\} \text{ para } i = 1, ..., n \end{gathered}$$

Se da un conjunto de n artículos con ganancias $c_i > 0$ y m recursos (mochilas) con capacidades $b_j > 0$. Cada artículo i consume una cantidad $a_{ji} \geq 0$ de cada mochila j. Las variables de decisión binarias x_{ij} indican qué elementos están seleccionados. El objetivo es elegir un subconjunto de elementos con el máximo beneficio total, de tal manera que los elementos seleccionados no deben exceder las capacidades de las mochilas; esto se expresa con las restricciones tipo mochila.

1.12.1. Aplicación

Para esta variante supongamos que Juan es un distribuidor de mercancía para 3 verdulerías cercanas, él dispone de 3 camionetas con la misma capacidad 193 kg tiene que planear su compra para 3 locales diferentes los cuales tienen pedidos especiales. Suponemos que todos los artículos conservan su peso (presentación). Los pedidos son:

- Para la verdulería 1 no existe pedido especial por lo que la capacidad se conserva en 193kg
- Para la verdulería 2 hicieron un pedido de
  - 1 caja de Aguacate Hass, ésta ocupa 20 kg de espacio.
  - 1 caja de Tomate Saladette, ocupa 26 kg de capacidad

  Con estos pedidos la capacidad de la camioneta será de 147 kg
- Para la verdulería 3 hicieron un pedido de
  - 1 costal de Papa Alpha, ocupan 50 kg de espacio.

Con este pedido la capacidad de la camioneta será de 143 kg

Con las variantes anteriores el modelo matemático es:

$$max\ z = 5x_1 + 7x_2 + 8x_3 + 9x_4 + 9x_5 + 5x_6 + 9x_7 + 10x_8 + 7x_9 + 9x_{10} + 8x_{11} + 6x_{12} + 5x_{13} + 9x_{14} + 7x_{15} + 9x_{16} + 10x_{17} + 7x_{18} + 6x_{19}$$

$$sujeto\ a$$

$$13x_1 + 13x_2 + 5x_3 + 13x_4 + 20x_5 + 20x_6 + 17x_7 + 5x_8 + 5x_9 + 17x_{10} + 17x_{11} + 17x_{12} + 18x_{13} + 35x_{14} + 8x_{15} + 8x_{16} + 10x_{17} + 3x_{18} + 3x_{19} \leq 193$$

$$13x_1 + 13x_2 + 5x_3 + 13x_4 + 20x_5 + 20x_6 + 17x_7 + 5x_8 + 5x_9 + 17x_{10} + 17x_{11} + 17x_{12} + 18x_{13} + 35x_{14} + 8x_{15} + 8x_{16} + 10x_{17} + 3x_{18} + 3x_{19} \leq 147$$

$$13x_1 + 13x_2 + 5x_3 + 13x_4 + 20x_5 + 20x_6 + 17x_7 + 5x_8 + 5x_9 + 17x_{10} + 17x_{11} + 17x_{12} + 18x_{13} + 35x_{14} + 8x_{15} + 8x_{16} + 10x_{17} + 3x_{18} + 3x_{19} \leq 143$$

$$x_i = \{0, 1\} \text{ para } i = 1, \ldots, 19$$

Algunos de los algoritmos para resolver este tipo de problemas son:

- Método de búsqueda binaria diferencial.
- Algoritmo híbrido (búsqueda tabú y programación lineal)

Artículos referentes al problema de la mochila multidimensional:

- *Weingartner, H. M., & Ness, D. N. (1967). Methods for the solution of the multidimensional 0/1 knapsack problem. Operations Research, 15(1), 83-103.*
- *Puchinger, J., Raidl, G. R., & Pferschy, U. (2010). The multidimensional knapsack problem: Structure and algorithms. INFORMS Journal on Computing, 22(2), 250-265.*

Algunas otras variantes del problema de la mochila son:

1.13. Problema de la mochila multicriterio

Un problema de optimización multicriterio es un problema que tiene múltiples criterios a optimizar, por ejemplo en el problema de la mochila lo que deseamos optimizar es el beneficio obtenido por artículos incluidos, pero si además queremos minimizar volumen.
Los datos del problema son los siguientes:

- c_i beneficio obtenido por unidad que se incluye del articulo del tipo i.
- v_i Volumen que ocupa por unidad que se incluye del artículo i
- b capacidad de la mochila
- n número de artículos

Entonces el modelo matemático con dos objetivos a optimizar es:

$$maxz = \Sigma_{i=1}^{n} c_i x_i;\ min\ w = \Sigma_{i=1}^{n} v_i x_i$$
$$sujeto\ a$$
$$\Sigma_{i=1}^{n} a_i x_i \leq b$$
$$x_i = \{0, 1\} \text{ para } i = 1, ..., n$$

1.14. Problema de la mochila con desplazamiento (collapsing)

El Problema de la Mochila 0-1 con desplazamiento es un problema no lineal, puesto que el tamaño de la mochila es una función decreciente que depende del número de artículos incluidos.
La no linealidad del problema radica en el lado derecho de la restricción.
Este planteamiento consiste en una partición de los artículos y cada partición tienen asignada una capacidad de la mochila, dicha capacidad se obtiene por medio de una función $h : R \to R$
Este problema se puede formular con el siguiente modelo matemático:

$$max\ z = \Sigma_{i=0}^{n} c_i x_i$$
$$sujeto\ a$$
$$\Sigma_{i=0}^{n} a_i x_i \leq h\Sigma_{i=0}^{n} x_i$$
$$x_i = \{0, 1\} \text{ para } i = 1, ..., n$$
$$h : R \to R$$

Una de las aplicaciones a este modelo es en comunicaciones por satélite. Cuando una banda de comunicaciones en particular tiene sólo un usuario (se envía acceso múltiple por división de frecuencia) se puede utilizar toda la banda. Sin embargo, las transmisiones múltiples en la banda requieren de particiones de intervalos entre las porciones de la banda asignada a cada usuario. Estas particiones evitan problemas de conversación cruzada, pero también reducen la capacidad de comunicación efectiva de la banda. Por lo tanto, dada una limitación de capacidad, el problema que los clientes pueden permitir en su banda de comunicaciones puede ser manejado con la formulación anterior.

- c_i toma el valor de cada cliente (generalmente los ingresos generados)
- a_i el tamaño del ancho de banda que cada cliente requiere
- h la capacidad de la banda en función del número de usuarios en el sistema.

Aunque el problema fue motivado por una aplicación específica, puede generalizarse. Otras aplicaciones en esta problema son:

- El diseño de un centro comercial en el que se debe seleccionar tanto el tipo como el tamaño de la tienda y la carga de artículos frágiles en un camión.
- Problemas de "multiplicidad". Un ejemplo es llevar una pelota de baloncesto (relativamente fácil). Sin embargo, aunque con menor volumen total, requeriría gran destreza para llevar cincuenta canicas. Así, la capacidad se ve afectada no sólo por el tamaño sino también por el número de elementos elegidos.

- Otro ejemplo de un problema de este tipo son las operaciones de la computadora en un entorno de tiempo compartido. Con un solo usuario en el sistema, el sistema puede usarse continuamente. Sin embargo, supongamos que dos o más usuarios están activos simultáneamente. Entonces el tiempo se debe gastar no sólo para encender a un usuario apagado y para traer otro encendido sino también para no perder de vista a cada uno de los usuarios que están en el sistema. A medida que aumenta el número de usuarios, disminuye la capacidad de procesamiento efectivo de la computadora. De aquí surge una pregunta: ¿Qué clientes se deben permitir en la computadora para optimizar los beneficios?

Primer artículo donde se mencionó el problema de la mochila 0-1 con desplazamiento

Posner, M. E., & Guignard, M. (1978). The collapsing 0-1 knapsack problem. Mathematical Programming, 15(1), 155-161.

Capítulo 2

Algoritmos exactos

Se conocen como algoritmos exactos a los algoritmos que proporcionan la solución óptima del problema (si existe), una de sus características es que cuentan con una prueba de optimalidad, ésta nos indica en qué momento se debe detener el algoritmo porque ya se encontró el óptimo o porque dicho óptimo no existe.

Como se ha mencionado anteriormente, la mayoría de los problemas de programación lineal entera son difíciles de resolver, esto quiere decir que existen instancias del problema para los cuales al aplicar el algoritmo exacto nos tardaríamos mucho tiempo para obtener la solución óptima, a pesar de esto, existen algunas instancias (pequeñas) de los problemas de optimización para los cuales el tiempo para resolverlos es relativamente rápido, al implementar los algoritmos a dichas instancias ayuda a mejorar el tiempo de ejecución mediante cotas o criterios para no realizar tantas iteraciones para llegar al óptimo.

Algunos de los métodos exactos son; Planos de corte, Ramificación y acotamiento, Programación dinámica, Ramificación y corte.

En este capítulo se explicarán:

- Ramificación y acotamiento
- Programación Dinámica

2.1. Ramificación y acotamiento

Este método empieza con la solución de la relajación PL del problema entero, ésta proporciona una cota superior en el valor de z (para el caso de un problema de maximización), si la solución encontrada es entera, entonces, será la óptima, en caso contrario será una cota superior para la función objetivo y se utilizará como prueba de optimalidad para las soluciones que se encuentren al desarrollar el algoritmo, además nos ayuda a etiquetar subproblemas como no prometedores (por ser una cota superior), a partir de la relajación PL se crean subproblemas que se derivan al dividir la región factible quitando las soluciones que no nos interesan, es decir, las soluciones no enteras; para dividir la región factible se utilizan restricciones nuevas (restricciones de acotamiento en la región factible) para tratar de hacer factible (entera) la solución de estos subproblemas.

Para explicar este procedimiento utilizaremos la región factible de la relajación PL de un problema entero y utilizamos los siguientes pasos:

1. Sean P un problema de PLE y P_1 la relajación PL de P, recordemos la contención entre las regiones factibles: sean Fp, F_{P_1} las regiones factibles de P y P_1 respectivamente, como se muestra en la figura:

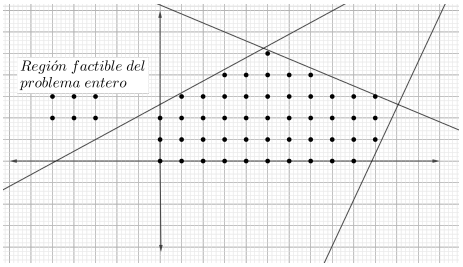


Figura 2.1: *F_p Región Factible del problema entero.*

Fuente: Elaboración propia utilizando Geogebra 5 (2016)

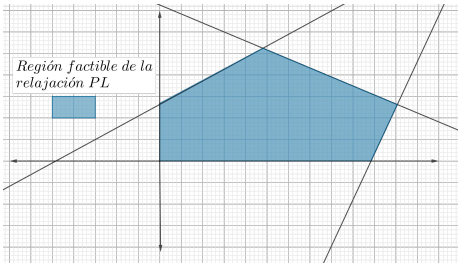


Figura 2.2: *F_{P_1} Región Factible de la relajación PL de P.*

Fuente: Elaboración propia utilizando Geogebra 5 (2016)

   Entonces $Fp \subset F_{P_1}$

2. Se resuelve P_1, si todas las entradas de la solución son enteras, terminamos, la solución obtenida es la óptima, en otro caso ir a 3

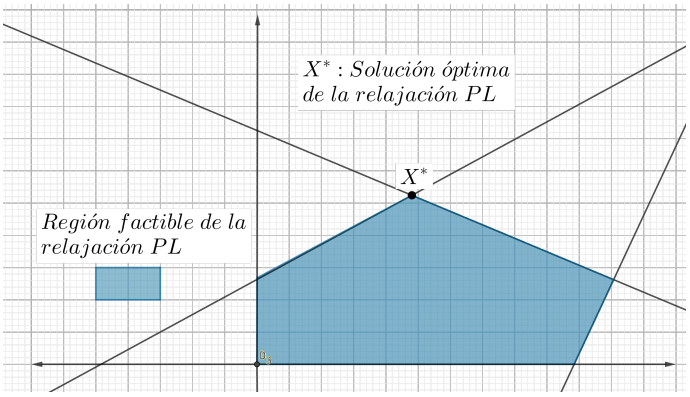


Figura 2.3: *Solución de la relajación PL*

Fuente: Elaboración propia utilizando Geogebra 5 (2016)

   Supongamos que la solución del problema relajado es $X^* = (4.81, 5.24)$, como tienen entradas no enteras, entonces, ir al paso 3

3. Si es más de una entrada no entera, se escoge una, sea x_i, a partir de ella se crean dos subproblemas P_2 y P_3 con las siguientes restricciones

$$x_i \leq [x_i], \text{ para } P_2$$
$$x_i \geq [x_i] + 1, \text{ para } P_3$$

donde $[x_i]$ es la parte entera de x_i.
Estas irán acotando la región factible de los subproblemas.
En el ejemplo, las dos variables son no enteras por lo que se escoge indistintamente alguna, sea x_1, entonces las restricciones de acotamiento asociadas a esta variable son:

$$x_1 \leq [4.81], \text{ para } P_2$$
$$x_1 \geq [4.81] + 1, \text{ para } P_3$$

es decir:

$$x_1 \leq 4, \text{ para } P_2$$
$$x_1 \geq 5, \text{ para } P_3$$

Cada subproblema P_2 y P_3 tienen su respectiva región factible F_{P_2} y F_{P_3}, existen soluciones descartadas por ser factibles para la relajación PL pero no para el problema entero, ya que tiene valores no enteros, por lo que no afecta eliminarlos de consideración.
Esto se muestra en la siguiente figura.

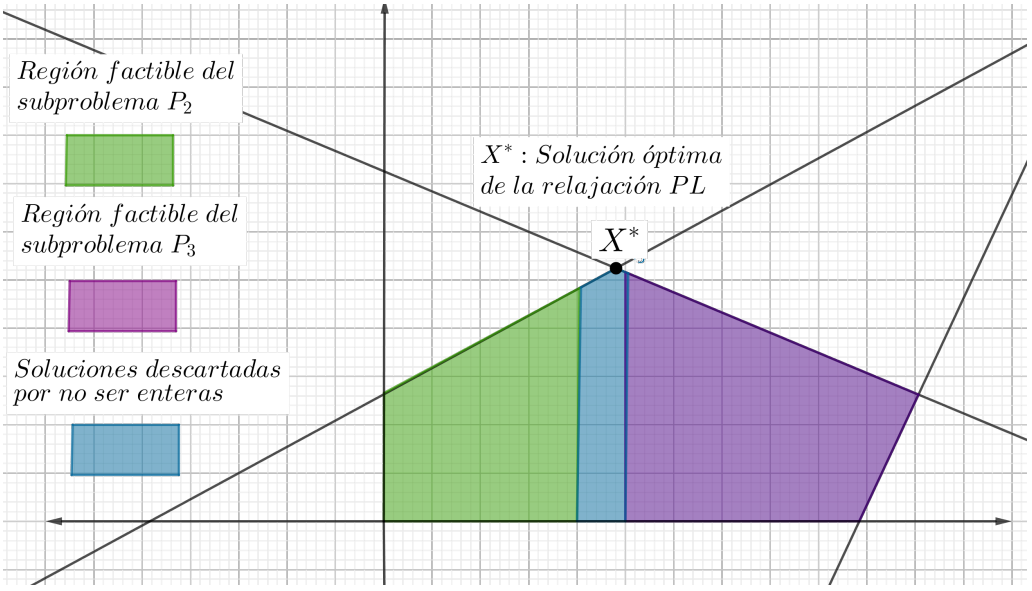


Figura 2.4: *Región Factible de los Subproblemas P_2 y P_3*

Fuente: Elaboración propia utilizando Geogebra 5 (2016)

4. Se resuelve cada uno de los subproblemas, Sean X_2 *y* X_3 las soluciones de P_2 *y* P_3 respectivamente,
Si las soluciones encontradas X_2 y X_3 no son enteras entonces se escoge uno de los subproblemas P_2 o P_3 para proceder de la misma manera que con P_1, este procedimiento continua hasta encontrar una solución óptima para el problema

P.

Cabe señalar que todos los subproblemas $P_{i's}$ tienen la misma función objetivo que P y las mismas restricciones, es decir, el mismo modelo matemático de la relajación PL de P, la diferencia es que, a los $P_{i's}$ se les aumentan algunas restricciones de ramificación para acotar la región factible, en otras palabras las restricciones se heredan y solo se aumentarán las nuevas restricciones que acotan más a la región factible para tratar de movernos a la región factible del problema entero.

Observación 8 *El valor óptimo de cada problema es una cota superior del valor óptimo de los subproblemas generados a partir de él, es decir, la función objetivo asociada a cada nodo es una cota superior de los nodos hijos (subproblemas que se generan a partir de él) pues la región factible de éstos está contenida en la del primero.*
Análogamente para problemas de minimización, el valor de la función objetivo de cada nodo es una cota inferior de los nodos hijos.

Todos los pasos de la ramificación y acotamiento se pueden resumir en un árbol donde cada nodo representa un subproblema (ramificación) y cada arista tiene asociada una restricción de acotamiento, esto se ilustra con la siguiente figura:

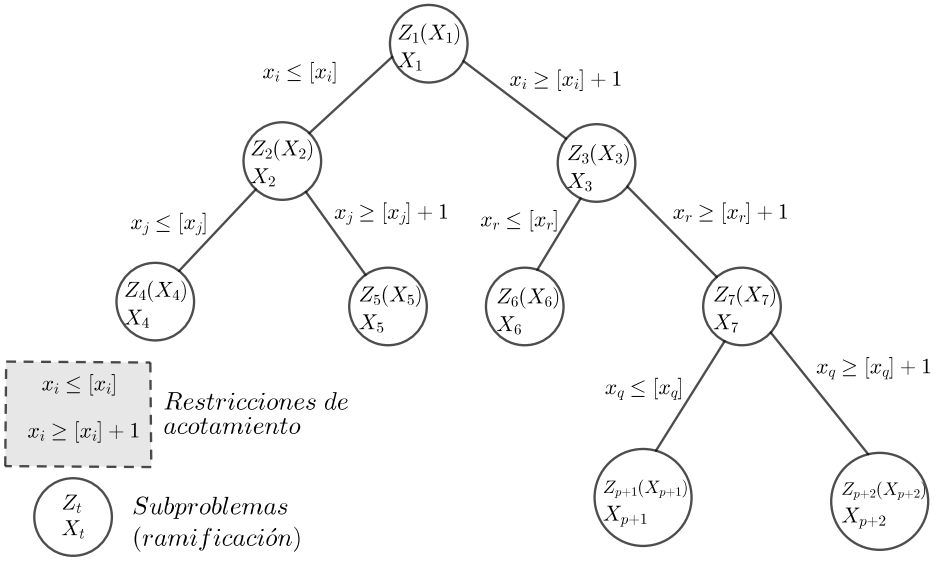


Figura 2.5: *Representación de los subproblemas mediante un árbol*

Fuente: Elaboración propia utilizando Geogebra 5 (2016)

Ilustrando la observación anterior si P es un problema de maximización:

- De la rama izquierda:
  $Z_1(X_1) \geq Z_2(X_2) \geq Z_4(X_4)$ *y* $Z_1(X_1) \geq Z_2(X_2) \geq Z_{5(}X_5)$

- De la rama derecha:
  $Z_1(X_1) \geq Z_3(X_3) \geq Z_6(X_6)$ *y* $Z_1(X_1) \geq Z_3(X_3) \geq Z_7(X_7)$

2.1.1. Aspectos claves del método de ramificación y acotamiento

Si utilizamos el método de ramificación y acotamiento, podemos "eliminar" algunos nodos en donde ya se llegó o no llegaríamos a una solución candidata u óptima y de esta manera se reduce el número de subproblemas a resolver.
No es necesario ramificar con respecto a un subproblema cuando:

1. El subproblema no es factible, es decir, que no cumpla alguna restricción del problema.

2. El subproblema proporciona una solución en la cual todas la variables tienen valores enteros, es decir, es una solución candidata para el problema entero.

3. El valor óptimo de la función objetivo del subproblema es menor al valor de la función objetivo de una solución candidata, es decir, el mejor candidato para z hasta este momento; a dicho nodo se etiquetará como nodo no prometedor. (Para el caso de un problema de maximización)

Con estos puntos se reduce el número de nodos en el árbol.

2.1.2. Algoritmo de ramificación y acotamiento (Land-Doig)

Existen muchas variantes de este algoritmo, en este trabajo explicaremos el que crea dos nodos por cada nodo pendiente, es decir, si la solución de un nodo X_t tiene una componente x_j que no es entera, al primer nodo que se crea se le asigna la restricción $x_j \leq [x_j]$ y al segundo $x_j \geq [x_j] + 1$; cada problema es resuelto. El procedimiento termina cuando la lista de nodos pendientes sin etiquetar es vacía. [Prawda W (1976)].
Propósito: Obtener la solución óptima de un problema de PLE (P).

1. **Solución del problema relajado:** Se obtiene la relajación PL de P, se obtienen el óptimo, sea X_1 dicha solución y $z_1 = z(X_1)$ la evaluación de X_1 en la función objetivo, si $x_j \in Z \ \forall j$ terminamos, la solución encontrada es óptima, si no, ir al paso 2.

2. **Ramificación** Sea X_i la solución de un subproblema que tiene una componente no entera, digamos x_j, se crean dos nodos; al problema correspondiente al nodo $l + 1$ se le agrega la restricción $x_j \leq [x_j]$ y al problema del segundo nodo $l + 2$ se le agrega la restricción $xj \geq [x_j] + 1$, donde l es el número del último nodo. Como se observa en la figura:

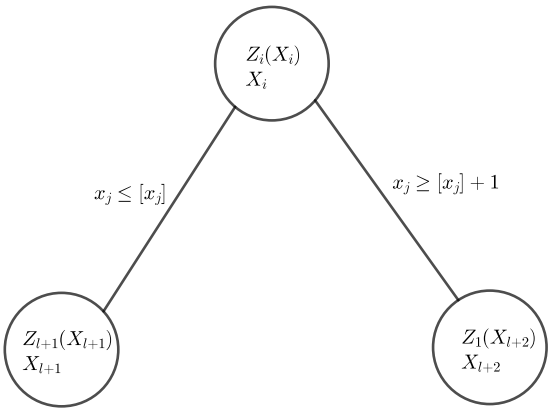


Figura 2.6: *Restricciones de Ramificación*

Fuente: Elaboración propia utilizando Geogebra 5 (2016)

Una vez resueltos los problemas de dichos nodos ir al paso 3:

3. **Etiquetado**

   a) Si la solución encontrada es entera dicho nodo es etiquetado como **Solución candidata**.

   b) Si $z(X_r)$ es menor que $z(X_k)$, para algún nodo X_k etiquetado como *Solución candidata*, entonces el nodo r será etiquetado como **Nodo no prometedor**.

   c) Si la solución encontrada no es factible (ya que no cumple con alguna restricción del problema) a este nodo se le asignará la etiqueta de **Solución no factible**.

   Después de haber etiquetado todos los nodos posibles ir al paso 4)

4. **Inspección de nodos**

   a) Si todos los nodos están etiquetados y no existe alguno que tenga la etiqueta de *Solución candidata* entonces terminamos, el problema entero no tiene solución.

   b) Si todos los nodos están etiquetados y existen alguno o algunos nodos etiquetados con *Solución candidata* ir al paso 5).

   c) Si existen nodos sin etiquetar, esto quiere decir que se puede seguir iterando de estos nodos, es decir, no se ha encontrado del óptimo, ir al paso 2).

5. **Solución óptima** El valor óptimo de la función objetivo es

$$z(X^*) = max\{z(X_i)|X_i \text{ es un nodo etiquetado con Solución candidata}\}$$

   la solución óptima es X^* .

Un punto clave del método es saber escoger el subproblema del que se seguirá ramificando, si es único el nodo que no tiene etiqueta, se ramificará de éste, pero si son varios nodos sin etiqueta se puede elegir dicho subproblema con alguno de los siguientes criterios:

- Por profundidad: Esto es escoger una de las dos ramas, izquierda o derecha, se debe ramificar siempre de ese lado hasta etiquetar a todos los nodos hijos.

- Mejor cota: Cuando se termine una iteración se elige el nodo con mejor valor de la función objetivo y se ramificará a partir de él.

- Al azar: Se ramificará del nodo que mejor nos parezca.

En realidad no existe alguna prueba que nos diga cual criterio es más eficiente y de aquí la dificultad de resolver un problema con este método, incluso se pueden combinar los criterios, es decir, podemos empezar por profundidad y después escoger el que tenga mejor cota.

2.1.3. Ramificación y Acotamiento aplicado a un problema de programación lineal entera

Para explicar el algoritmo utilizaremos un problema de programación lineal entera.

Ejemplo 5 *Consideremos el siguiente problema:*

$$
\begin{gathered}
max\ z = 5x_1 + 2x_2 \\
sujeto\ a \\
3x_1 + x_2 \leq 12 \\
x_1 + x_2 \leq 5 \\
x_1, x_2 \geq 0 \\
x_1, x_2 \in Z
\end{gathered}
$$

La región factible del problema entero es:

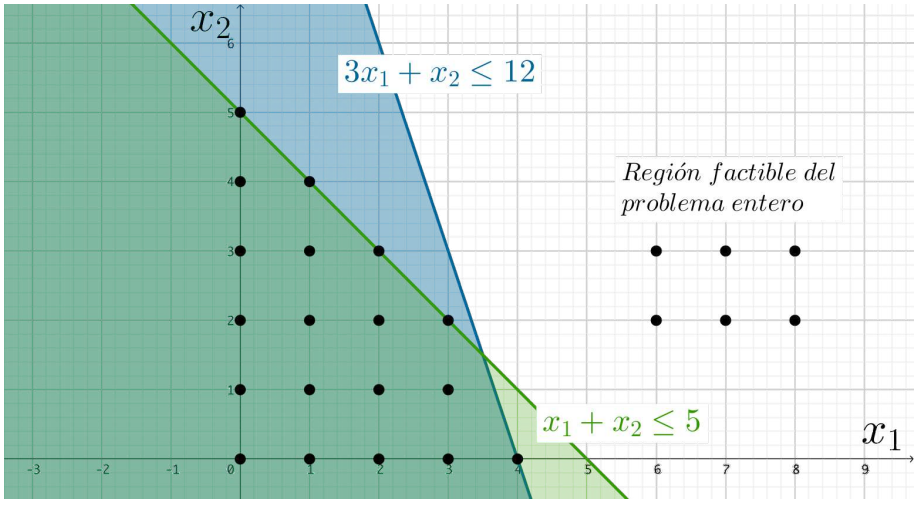


Figura 2.7: *Región factible del problema entero*

Fuente: Elaboración propia utilizando Geogebra 5 (2016)

Relajamos el problema y obtenemos el problema P_1

$$P_1 \begin{cases} max\ z = 5x_1 + 2x_2 \\ \quad sujeto\ a \\ 3x_1 + x_2 \leq 12 \\ x_1 + x_2 \leq 5 \\ x_1, x_2 \geq 0 \end{cases}$$

La región factible del problema y el óptimo (utilizando Lingo 17*) son:*

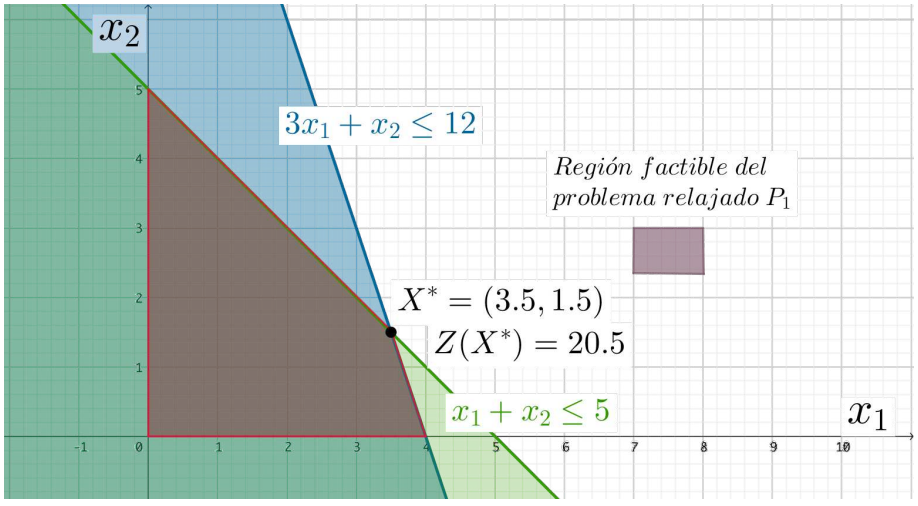


Figura 2.8: *F_p y Solución óptima del problema relajado*

Fuente: Elaboración propia utilizando Geogebra 5 (2016)

Como $x_1 \notin Z$ *y* $x_2 \notin Z$ *escogemos alguna de las dos variables de decisión del problema para empezar a acotar la región factible, sea* x_2*; entonces, las restricciones de cada subproblema son:*

$$x_2 \leq [1.5], \textit{ para } P_2$$
$$x_2 \geq [1.5] + 1, \textit{ para } P_3$$

Quedando las restricciones:

$$x_2 \leq 1, \textit{ para } P_2$$
$$x_2 \geq 2, \textit{ para } P_3$$

Los subproblemas P_2 y P_3 con sus respectivas restricciones de acotamiento son:

$$P_2 \begin{cases} max\ z = 5x_1 + 2x_2 \\ sujeto\ \ a \\ 3x_1 + x_2 \leq 12 \\ x_1 + x_2 \leq 5 \\ x_2 \leq 1 \\ x_1, x_2 \geq 0 \end{cases} \qquad P_3 \begin{cases} max\ z = 5x_1 + 2x_2 \\ sujeto\ \ a \\ 3x_1 + x_2 \leq 12 \\ x_1 + x_2 \leq 5 \\ x_2 \geq 2 \\ x_1, x_2 \geq 0 \end{cases}$$

Las regiones factibles de cada subproblema son:

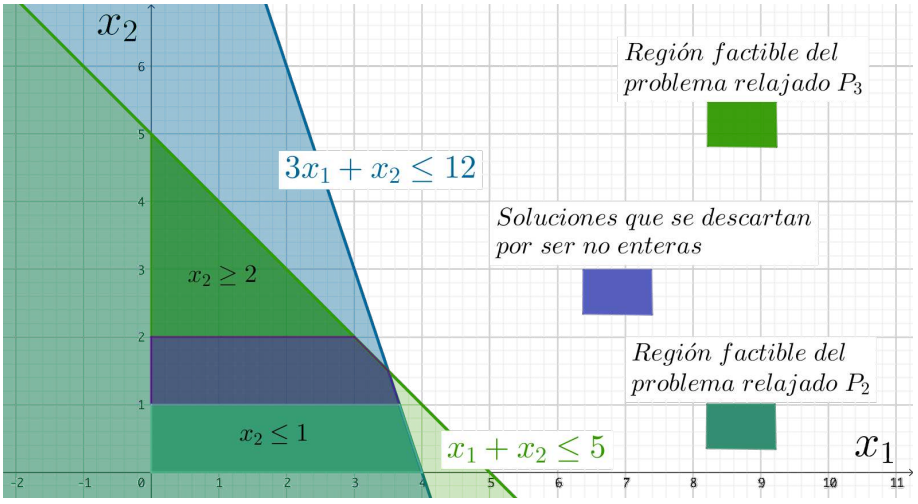


Figura 2.9: *Región factible de los subproblemas P_2 y P_3*

Fuente: Elaboración propia utilizando Geogebra 5 (2016)

Resolvemos con Lingo ambos problemas y obtenemos:

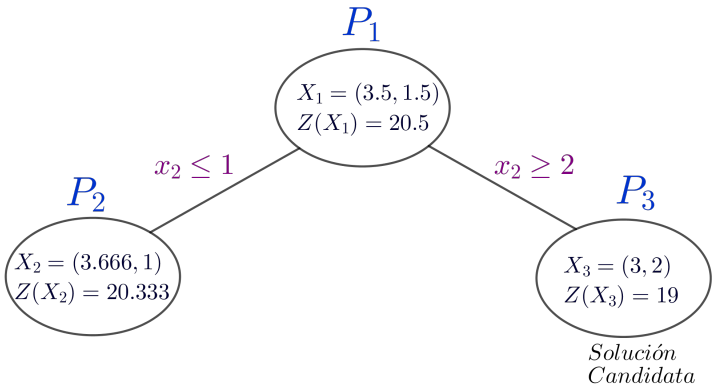


Figura 2.10: *Soluciones de los problemas P_2 y P_3*

Fuente: Elaboración propia utilizando Geogebra 5 (2016)

Observemos que en el problema P_3 tenemos una solución donde todas las entradas son enteras $X_3 = (3, 2)$ por esta razón se etiqueta como Solución Candidata, pero todavía no podemos decir que es la óptima ya que a pesar de que la solución de P_2 no es entera $X_2 = (3.666, 1)$ la función objetivo es mejor que la del problema P_3 $(z(X_2) = 20.333 > 19 = z(X_3))$ esto quiere decir que del problema P_2 podríamos encontrar una solución mejor que la obtenida en P_3; por lo que ramificaremos del problema P_2.

De la solución $X_2 = (3.666, 1)$ la única variable que no es entera es x_1 por lo que se acotará la región factible de P_2 utilizando a la variable x_1, eso es:

$$x_1 \leq [3.666], \text{ para } P_4$$
$$x_1 \geq [3.666] + 1, \text{ para } P_5$$

Quedando las restricciones:

$$x_1 \leq 3, \text{ para } P_4$$
$$x_2 \geq 4, \text{ para } P_5$$

Recordemos que las restricciones de P_2 se heredan y a los problemas hijos P_4 y P_5 se le agregan sus respectivas restricciones de acotamiento, éstas son:

$$P_4 \begin{cases} max\ z = 5x_1 + 2x_2 \\ sujeto\ \ a \\ 3x_1 + x_2 \leq 12 \\ x_1 + x_2 \leq 5 \\ x_2 \leq 1 \\ x_1 \leq 3 \\ x_1, x_2 \geq 0 \end{cases} \qquad P_5 \begin{cases} max\ z = 5x_1 + 2x_2 \\ sujeto\ \ a \\ 3x_1 + x_2 \leq 12 \\ x_1 + x_2 \leq 5 \\ x_2 \leq 1 \\ x_1 \geq 4 \\ x_1, x_2 \geq 0 \end{cases}$$

Las regiones factibles de cada subproblema son:

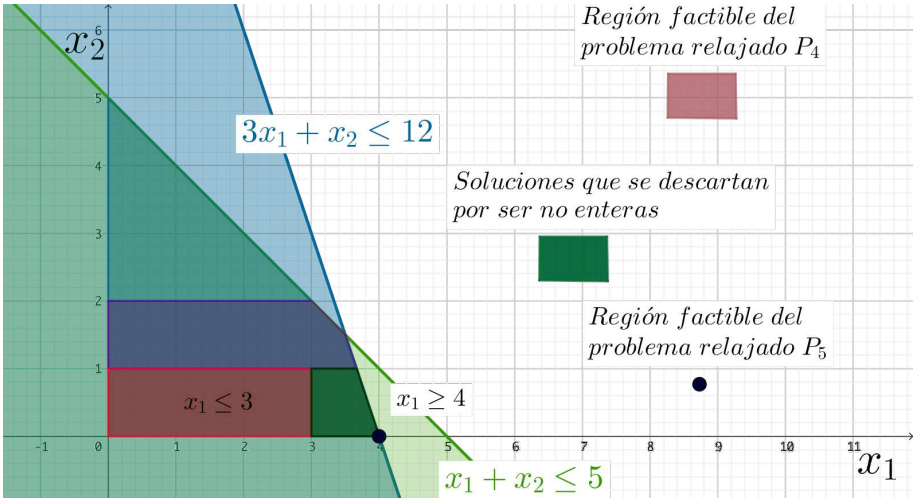


Figura 2.11: *Región factible de los subproblemas P_4 y P_5*

Fuente: Elaboración propia utilizando Geogebra 5 (2016)

Resolvemos con Lingo ambos problemas y obtenemos:

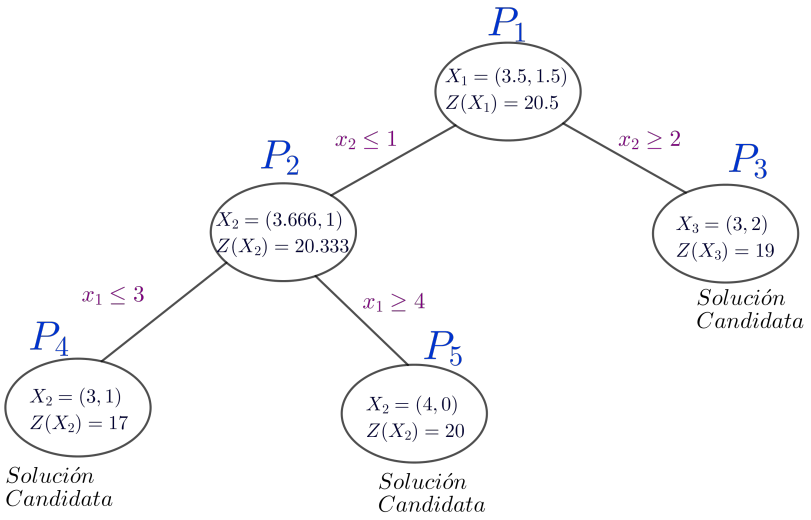


Figura 2.12: *Soluciones de los problemas P_4 y P_5*

Fuente: Elaboración propia utilizando Geogebra 5 (2016)

En esta última iteración observamos que todos los nodos (subproblemas) están etiquetados, entonces el óptimo lo obtenemos:

$$z(X^*) = max\{z(X_i)|X_i \text{ es un nodo etiquetado con Solución candidata}\}$$

y la solución óptima es X^*
Aplicando esto al problema:

$$z(X^*) = max\{19, 17, 20\} = 20$$

que es la solución asociada al problema P_5, por lo que $X^* = (4, 0)$ con $z(X^*) = 20$

Este método se puede aplicar a los problemas enteros como lo hicimos en el ejemplo anterior, pero existen algunos problemas combinatorios que tienen ciertas características que se tienen que aprovechar y se podría decir que tienen su propio método, pero la base es el algoritmo anterior. El problema del agente viajero y el problema de la mochila son de este tipo de problemas.

2.1.4. Ramificación y acotamiento para el problema de la mochila

Recordemos el problema de la mochila:

$$\begin{gathered} max\ z = \Sigma_{i=1}^{n} c_i x_i \\ sujeto\ a \\ \Sigma_{i=1}^{n} a_i x_i \leq b \\ x_i \in \{0,1\} \text{ para } i = 1, ..., n \text{ (Binario)} \\ x_i \in Z \text{ para } i = 1, ..., n \text{ (Genral)} \end{gathered}$$

Como solo tiene una restricción lineal es posible resolver el problema relajado de manera rápida, lo importante es meter a todos los artículos que quepan en la mochila tal que se maximice el beneficio, pero ¿qué artículos se deben meter primero?, para responder a esta pregunta es importante ordenar los artículos de acuerdo al beneficio pero sin olvidar el peso, es decir. ordenar los siguientes cocientes

$$\frac{c_i}{a_i} \qquad\qquad para\ i = 1, \ldots, n$$

La información que nos proporciona es el beneficio marginal por artículo incluido, si los ordenamos de manera decreciente, tendremos el artículo del que se obtiene mayor beneficio, esta información se utiliza de acuerdo al tipo de problema: binario, general o acotado.

Solución del problema relajado. (Un método para obtener la cota superior)

- **Problema general:** Supongamos que calculamos los cocientes $\frac{c_i}{a_i}$, los ordenamos en orden decreciente y el artículo del cual se obtiene mayor beneficio es el $j - esímo$, entonces la solución óptima de la relajación PL es

$$X_1 = \left(0, 0, \ldots, \frac{b}{a_j}, \ldots, 0, 0\right)$$

con

$$Z(X_1) = c_j \left(\frac{b}{a_j}\right)$$

Es decir, vamos a meter a la mochila lo más que se pueda del artículo que nos da mayor beneficio.

Ejemplo 6 *Consideremos la siguiente instancia del problema de la mochila:*

$$max\ z = 6x_1 + 16x_2 + 10x_3 + 20x_4$$
$$sujeto\ a$$
$$6x_1 + 14x_2 + 9x_3 + 17x_4 \leq 40$$
$$x_i \geq 0\ x_i \in Z\ para\ i = 1, 2, 3, 4$$

primero obtendremos la relajación PL, esto es omitir la restricción $x_i \in Z$ *para* $i = 1, 2, 3, 4$*, por lo tanto el problema relajado es:*

$$max\ z = 6x_1 + 16x_2 + 10x_3 + 20x_4$$
$$sujeto\ a$$
$$6x_1 + 14x_2 + 9x_3 + 17x_4 \leq 40$$
$$x_i \geq 0\ para\ i = 1, 2, 3, 4$$

para llegar a la solución óptima se deben obtener los cocientes $\frac{c_i}{a_i}$ *para* $i = 1, ..., 4$

Cuadro 2.1: *Beneficio marginal por artículo (problema general)*

Orden inicial	cociente $\frac{c_i}{a_i}$	nuevo orden
1º (x_1)	$\frac{6}{6} = 1$	4º
2º (x_2)	$\frac{16}{14} \approx 1.14$	2º
3^{er} (x_3)	$\frac{10}{9} \approx 1.11$	3^{er}
4º (x_4)	$\frac{40}{17} \approx 2.35$	1º

Como el artículo del que se obtiene mayor beneficio es el 4*, entonces la solución óptima del problema relajado es:* $x_4 = \frac{40}{17} = 2.35$ *y las demás entradas igual a cero, es decir,*

$$X_1 = (0, 0, 0, 2.35)$$

$$z(X_1) = 47.05$$

- **Problema acotado:** Supongamos que el nuevo orden de las variables es:

$$x_{1'}, x_{2'}, \ldots, x_{n'}$$

como alguna o todas las variables del problema tienen la restricción $x_j \leq d_j$ entonces caemos en 2 casos para obtener la solución de la relajación PL

  - Si la cota del artículo que nos genera mayor beneficio es mayor o igual que $\frac{b}{a_{1'}}$, es decir, $\frac{b}{a_{1'}} \leq d_{1'} \Rightarrow X_1 = \left(\frac{b}{a_{1'}}, 0, \ldots, 0, 0\right)$ y $Z(X_1) = c_{1'}\left(\frac{b}{a_{1'}}\right)$
  - Si la cota del artículo que nos genera mayor beneficio es menor que $\frac{b}{a_{1'}}$, es decir, $\frac{b}{a_{1'}} > d_{1'}$ volvemos a caer en dos casos:

- Si la cota del siguiente artículo que nos genera mayor beneficio es mayor o igual que $\frac{b-d_{1'}}{a_{2'}}$, es decir, $\frac{b-d_{1'}}{a_{2'}} \leq d_{2'} \Rightarrow X_1 = \left(d_{1'}, \frac{b-d_{1'}}{a_{2'}}, 0, \ldots, 0, 0\right)$ y $Z(X_1) = c_{1'}d_{1'} + c_{2'}\left(\frac{b-d_{1'}}{a_{2'}}\right)$
- Si la cota del artículo que nos genera mayor beneficio es menor que $\frac{b-d_{1'}}{a_{2'}}$,es decir, $\frac{b-d_{1'}}{a_{2'}} > d_{2'}$ volvemos a caer en dos casos, este procedimiento se repite hasta saturar la capacidad de la mochila.

Sin pérdida de generalidades podemos concluir que la solución de la relajación PL de un problema acotado de la mochila es:

$$X_1 = \left(d_{1'}, d_{2'} \ldots, d_{t'}, \left(\frac{1}{a_{(t+1)'}}\right) * \left(b - \Sigma_{i=1'}^{t'} a_i\right), \ldots, 0\right)$$

donde t' es el índice más pequeño entre 1 *y* n el cual satisface $\Sigma_{i=1'}^{t'} d_i a_i < b$ y $\Sigma_{i=1'}^{(t+1)'} d_i a_i > b$

Ejemplo 7 *Consideremos el siguiente problema:*

$$\begin{gathered} max\ z = 18x_1 + 14x_2 + 8x_3 + 4x_4 \\ sujeto\ a \\ 15x_1 + 12x_2 + 7x_3 + 4x_4 \leq 33 \\ x_1 \leq 2\ x_2 \leq 2\ x_3 \leq 3\ x_4 \leq 2 \\ x_i \geq 0\ x_i \in Z\ para\ i = 1, 2, 3, 4 \end{gathered}$$

primero obtendremos la relajación PL, esto es omitir la restricción $x_i \in Z$ *para* $i = 1, 2, 3, 4$*, por lo tanto el problema relajado es:*

$$\begin{gathered} max\ z = 18x_1 + 14x_2 + 8x_3 + 4x_4 \\ sujeto\ a \\ 15x_1 + 12x_2 + 7x_3 + 4x_4 \leq 33 \\ x_1 \leq 2\ x_2 \leq 2\ x_3 \leq 3\ x_4 \leq 2 \\ x_i \geq 0\ para\ i = 1, 2, 3, 4 \end{gathered}$$

para llegar a la solución óptima se deben obtener los cocientes $\frac{c_i}{a_i}$ *para* $i = 1, ..., 4$

Cuadro 2.2: *Beneficio marginal por artículo (problema acotado)*

Orden inicial	cociente $\frac{c_i}{a_i}$	nuevo orden
1^o (x_1)	$\frac{18}{15} = 1.2$	1^o
2^o (x_2)	$\frac{14}{12} \approx 1.16$	2^o
3^{er} (x_3)	$\frac{8}{7} \approx 1.14$	3^{er}
4^o (x_4)	$\frac{4}{4} = 1$	4^o

Como el orden de las variables no cambio, solo vamos a cuidar que la solución cumpla con la cota de cada variable, entonces la solución óptima del problema relajado es:

$$X_1 = (2, 0.25, 0, 0)$$
$$z(X_1) = 39.5$$

- **Problema binario:** Notemos que para resolver la relajación PL de este problema es análogo que el caso del problema acotado solo debemos suponer que la cota para todas las variables es 1, es decir

$$x_i \leq 1 \; \forall i$$

$$X_1 = \left(1, 1. \ldots, 1, \left(\frac{1}{a_{(t+1)'}}\right) * \left(b - \Sigma_{i=1'}^{t'} a_i\right), \ldots, 0\right)$$

donde t' es el índice más pequeño entre 1 *y* n el cual satisface $\Sigma_{i=1'}^{t'} a_i < b$ y $\Sigma_{i=1'}^{(t+1)'} a_i > b$

Ejemplo 8 *Consideremos el siguiente problema:*

$$max\ z = 50x_1 + 50x_2 + 64x_3 + 46x_4 + 50x_5 + 5x_6$$
$$sujeto\ a$$
$$56x_1 + 59x_2 + 80x_3 + 64x_4 + 75x_5 + 17x_6 \leq 190$$
$$x_i \in \{0, 1\}\ para\ i = 1, \ldots, 6$$

primero obtendremos la relajación PL, esto es omitir la restricción $x_i \in \{0, 1\}$ *para* $i = 1, \ldots, 6$*, por lo tanto el problema relajado es:*

$$max\ z = 50x_1 + 50x_2 + 64x_3 + 46x_4 + 50x_5 + 5x_6$$
$$sujeto\ a$$
$$56x_1 + 59x_2 + 80x_3 + 64x_4 + 75x_5 + 17x_6 \leq 190$$
$$0 \leq x_i \leq 1\ para\ i = 1, \ldots, 6$$

para llegar a la solución óptima se deben obtener los cocientes $\frac{c_i}{a_i}$ *para* $i = 1, \ldots, 6$

Cuadro 2.3: *Cocientes* $\frac{c_i}{a_i}$ *ordenados (problema binario)*

Orden inicial	cociente $\frac{c_i}{a_i}$	nuevo orden
1º (x_1)	$\frac{50}{56} \approx 0.89$	1º
2º (x_2)	$\frac{50}{59} \approx 0.84$	2º
3^{er} (x_3)	$\frac{64}{80} \approx 0.8$	3º
4º (x_4)	$\frac{46}{64} \approx 0.71$	4º
5º (x_5)	$\frac{50}{75} \approx 0.66$	5º
6º (x_6)	$\frac{5}{17} \approx 0.29$	6º

Tomando en cuenta el nuevo orden de las variables y cuidando que la solución cumpla con la cota de cada variable, además de la restricción de capacidad, la solución óptima del problema relajado es:

$$X_1 = (1, 1, 0.93, 0, 0, 0)$$

$$z(X_1) = 160$$

Aplicación del método

Ahora aplicaremos el algoritmo de ramificación y acotamiento para resolver cada uno de los problemas.

- **Problema general**
  Para resolver este problema se ramificará con las siguientes restricciones para acotar a la región factible.

$$x_i \leq [x_i] \; para \; P_{l+1}$$
$$x_i \geq [x_i] + 1 \; para \; P_{l+2}$$

Tomando en cuenta el problema del ejemplo 6, la relajación PL es:

$$\begin{gathered} max \; z = 6x_1 + 16x_2 + 10x_3 + 20x_4 \\ sujeto \; a \\ 6x_1 + 14x_2 + 9x_3 + 17x_4 \leq 40 \\ x_i \geq 0 \; para \; i = 1, 2, 3, 4 \end{gathered}$$

Y la solución óptima es:

$$X_1 = (0, 0, 0, 2.35)$$

$$z(X_1) = 47.05$$

Como la solución del problema relajado no es entera, empezamos a ramificar acotando la variable x_4 con las siguientes restricciones:

$$x_4 \leq [2.35] \; para \; P_2$$

$$x_4 \geq [2.35] + 1 \; para \; P_3$$

quedando:

$$x_4 \leq 2 \; para \; P_2$$

$$x_4 \geq 3 \; para \; P_3$$

Dichos problemas se resuelven utilizando los cocientes ordenados:
Iteración 1

- **Problema 2** Lo nombramos 2 ya que la relajación PL es el 1

$$max\ z = 6x_1 + 16x_2 + 10x_3 + 20x_4$$
$$sujeto\ a$$
$$6x_1 + 14x_2 + 9x_3 + 17x_4 \leq 40$$
$$x_4 \leq 2$$
$$x_i \geq 0\ para\ i = 1, 2, 3, 4$$

$\Rightarrow\ X_2 = (0, 0.42, 0, 2)$ y $z(X_2) = 46.85$

- **Problema 3**

$$max\ z = 6x_1 + 16x_2 + 10x_3 + 20x_4$$
$$sujeto\ a$$
$$6x_1 + 14x_2 + 9x_3 + 17x_4 \leq 40$$
$$x_4 \geq 3$$
$$x_i \geq 0\ para\ i = 1, 2, 3, 4$$

$\Rightarrow\ X_3 = (0, 0, 0, 3)$
la cual es entera, pero no satisface la restricción de capacidad, por lo que este problema se etiquetará como problema (nodo) no factible y no se seguirá ramificando de él.

Está iteración se resume en el siguiente árbol:

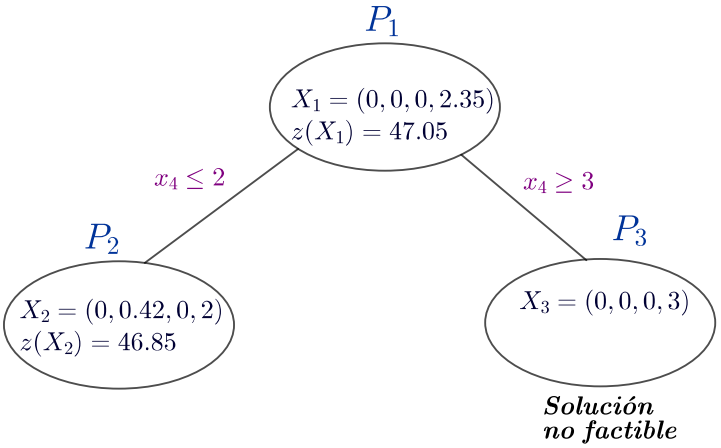


Figura 2.13: *Soluciones de los subproblemas P_2 y P_3*

Fuente: Elaboración propia utilizando Geogebra 5 (2016)

Como todavía no encontramos una solución entera factible, ramificamos del nodo 2 porque del nodo 3 ya no es posible.

Iteración 2

Las restricciones para acotar a la región factible son:

$$x_2 \leq [0.42]\ para\ P_4$$

$$x_2 \geq [0.42] + 1 \; para \; P_5$$

quedando:

$$x_2 \leq 0 \; es \; decir \; x_2 = 0 \; para \; P_4$$

$$x_2 \geq 1 \; para \; P_5$$

- **Problema 4** Hereda las restricciones del Problema 2 y aumentamos $x_2 = 0$

$$\begin{gathered} max \; z = 6x_1 + 16x_2 + 10x_3 + 20x_4 \\ sujeto \; a \\ 6x_1 + 14x_2 + 9x_3 + 17x_4 \leq 40 \\ x_4 \leq 2 \\ x_2 = 0 \\ x_i \geq 0 \; para \; i = 1, 2, 3, 4 \end{gathered}$$

$\Rightarrow \; X_4 = (0, 0, 0.66, 2)$ y $z(X_4) = 46.66$

- **Problema 5** También hereda las restricciones del nodo 2 y aumentamos $x_2 \geq 1$

$$\begin{gathered} max \; z = 6x_1 + 16x_2 + 10x_3 + 20x_4 \\ sujeto \; a \\ 6x_1 + 14x_2 + 9x_3 + 17x_4 \leq 40 \\ x_4 \leq 2 \\ x_2 \geq 1 \\ x_i \geq 0 \; para \; i = 1, 2, 3, 4 \end{gathered}$$

$\Rightarrow \; X_5 = (0, 1, 0, 1.52)$ y $z(X_5) = 46.58$
Está iteración se resume en el siguiente árbol:

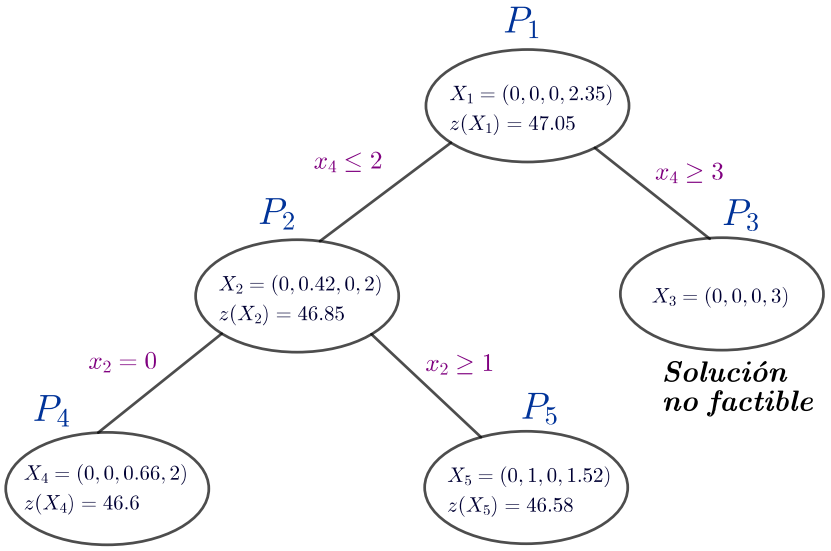


Figura 2.14: *Soluciones de los subproblemas P_4 y P_5*

Fuente: Elaboración propia utilizando Geogebra 5 (2016)

Nosotros vamos a escoger el que tiene la mejor cota. Recordemos que esto no nos garantiza que llegaremos más rápido al óptimo.
Iteración 3: Escogemos el nodo 4 para seguir ramificando
Las restricciones para acotar la región factible se asocian a la variable x_3, estas son:

$$x_3 \leq [0.66] \; para \; P_6$$
$$x_3 \geq [0.66] + 1 \; para \; P_7$$

quedando:

$$x_3 \leq 0 \; es \; decir \; x_2 = 0 \; para \; P_6$$
$$x_3 \geq 1 \; para \; P_7$$

generando los siguientes problemas:

- **Problema** 6 Hereda las restricciones del Problema 4 y aumentamos la restricción $x_3 = 0$

$$max \; z = 6x_1 + 16x_2 + 10x_3 + 20x_4$$
$$sujeto \; a$$
$$6x_1 + 14x_2 + 9x_3 + 17x_4 \leq 40$$
$$x_4 \leq 2$$
$$x_2 = 0$$
$$x_3 = 0$$
$$x_i \geq 0 \; para \; i = 1, 2, 3, 4$$

$\Rightarrow$ $X_6 = (1, 0, 0, 2)$ y $z(X_6) = 46$
Es una solución factible y entera por lo que este nodo se etiquetará como **Solución candidata**

- **Problema** 7 También hereda las restricciones del Problema 4 y aumentamos la restricción $x_3 \geq 1$

$$max\ z = 6x_1 + 16x_2 + 10x_3 + 20x_4$$
$$sujeto\ a$$
$$6x_1 + 14x_2 + 9x_3 + 17x_4 \leq 40$$
$$x_4 \leq 2$$
$$x_2 = 0$$
$$x_3 \geq 1$$
$$x_i \geq 0\ para\ i = 1, 2, 3, 4$$

$\Rightarrow$ $X_7 = (0, 0, 1, 1.82)$ y $z(X_7) = 46.47$
Está iteración se resume en el siguiente árbol:

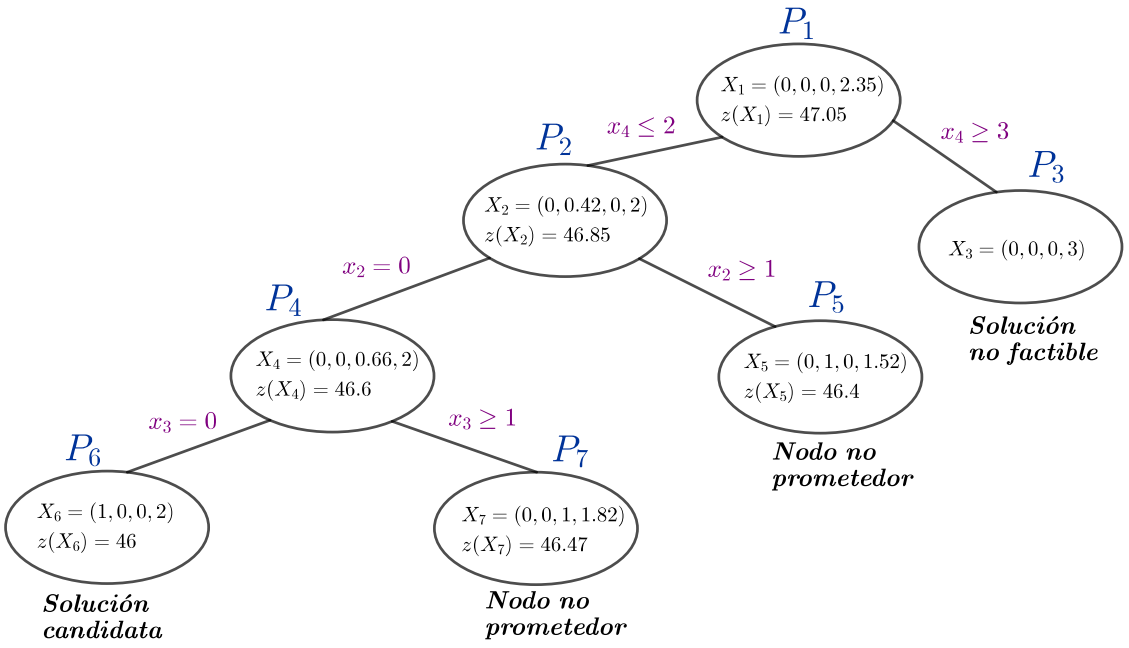


Figura 2.15: *Soluciones de los subproblemas P_6 y P_7*

Fuente: Elaboración propia utilizando Geogebra 5 (2016)

Encontramos una **Solución candidata** en el nodo 6 con $z(X_6) = 46$, esta información la tenemos que aprovechar ya que es una cota para z, con esto podemos etiquetar a los nodos 5 y 7 como **nodo no prometedor** ya que el valor de $z(X_5) = 46.4$ y $z(X_7) = 46.47$ son menores que 47 y como lo mencionamos anteriormente son cotas superiores de los Problemas que se generen a partir de él, en otras palabras, si llegáramos a encontrar otra solución candidata de

algún problema hijo de los nodos 5 o 7 a lo más el valor de z sería 46 (debido a que los coeficientes de z son enteros) la cual no mejoraría a la solución obtenida en el problema 6. Por lo tanto la solución óptima es:

$$X^* = (1, 0, 0, 2)$$

y

$$z(X^*) = 46$$

- **Problema acotado**
  Ahora aplicaremos ramificación y acotamiento para un problema acotado, para este tipo de problemas cambia la forma de ramificar ya que algunas o todas las variables tienen una cota d_j, entonces en lugar de generar solo dos subproblemas se generarán $d_j + 1$ subproblemas (para tomar en cuenta la alternativa $x_j = 0$), en otras palabras se generarán tantos subproblemas como alternativas tenga esta variable.
  La relajación PL de este problema es:

$$\begin{gathered} max\ z = 18x_1 + 14x_2 + 8x_3 + 4x_4 \\ sujeto\ a \\ 15x_1 + 12x_2 + 7x_3 + 4x_4 \leq 33 \\ x_1 \leq 2\ x_2 \leq 2\ x_3 \leq 3\ x_4 \leq 2 \\ x_i \geq 0 \text{ para } i = 1, 2, 3, 4 \end{gathered}$$

  La solución óptima es:

$$X_1 = (2, 0.25, 0, 0)$$

$$z(X_1) = 39.5$$

  Iteración 1 Como la solución no es entera, se empezará a ramificar sobre la variable x_2 que no es entera, además como $x_2 \leq 2$, entonces se crean 3 problemas hijos, cada uno de ellos con su respectiva restricción para acotar a la variable, estas son:

$$\begin{gathered} x_2 = 0\ para\ P_2 \\ x_2 = 1\ para\ P_3 \\ x_2 = 2\ para\ P_4 \end{gathered}$$

Los problemas a resolver son:

- **Problema** 2

$$\begin{gathered} max\ z = 18x_1 + 14x_2 + 8x_3 + 4x_4 \\ sujeto\ a \\ 15x_1 + 12x_2 + 7x_3 + 4x_4 \leq 33 \\ x_2 = 0 \\ x_1 \leq 2\ x_2 \leq 2\ x_3 \leq 3\ x_4 \leq 2 \\ x_i \geq 0 \text{ para } i = 1, 2, 3, 4 \end{gathered}$$

$\Rightarrow\ X_2 = (2, 0, 0.42, 0)$
y $z(X_2) = 39.42$

- **Problema** 3

$$\begin{gathered} max\ z = 18x_1 + 14x_2 + 8x_3 + 4x_4 \\ sujeto\ a \\ 15x_1 + 12x_2 + 7x_3 + 4x_4 \leq 33 \\ x_2 = 1 \\ x_1 \leq 2\ x_2 \leq 2\ x_3 \leq 3\ x_4 \leq 2 \\ x_i \geq 0 \text{ para } i = 1, 2, 3, 4 \end{gathered}$$

$\Rightarrow\ X_3 = (1.4, 1, 0, 0)$
y $z(X_3) = 39.2$

- **Problema** 4

$$\begin{gathered} max\ z = 18x_1 + 14x_2 + 8x_3 + 4x_4 \\ sujeto\ a \\ 15x_1 + 12x_2 + 7x_3 + 4x_4 \leq 33 \\ x_2 = 2 \\ x_1 \leq 1\ x_2 \leq 2\ x_3 \leq 3\ x_4 \leq 2 \\ x_i \geq 0 \text{ para } i = 1, 2, 3, 4 \end{gathered}$$

$\Rightarrow\ X_4 = (0.6, 2, 0, 0)$ y $z(X_4) = 38.8$

Cada problema es resuelto y esto se muestra en el siguiente árbol:

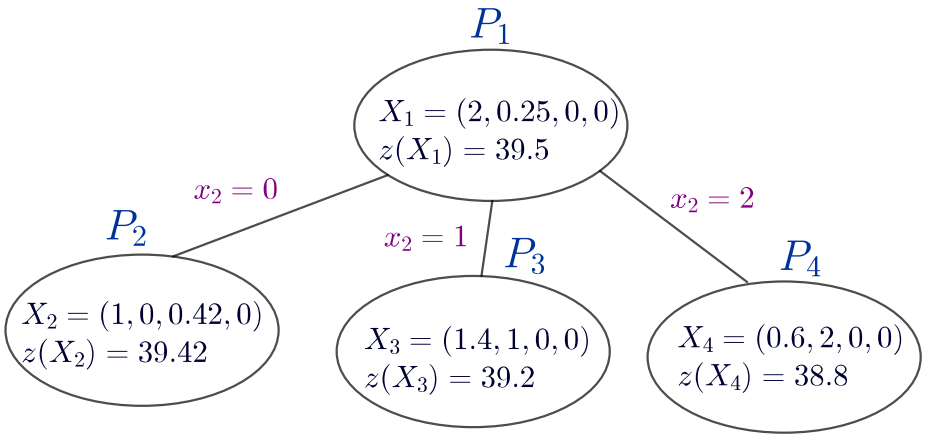


Figura 2.16: *Soluciones de los subproblemas P_2, P_3 y P_4*

Fuente: Elaboración propia utilizando Geogebra 5 (2016)

Iteración 2 Como ninguna solución es entera, seguimos ramificando y escogemos la mejor cota, es decir el problema 2.
Se ramificará con respecto a la variable x_3, las restricciones para acotar la región factible son:

$$x_3 = 0 \; para \; P_5$$
$$x_3 = 1 \; para \; P_6$$
$$x_3 = 2 \; para \; P_7$$
$$x_3 = 3 \; para \; P_8$$

Los problemas a resolver son:

- **Problema** 5

$$max \; z = 18x_1 + 14x_2 + 8x_3 + 4x_4$$
$$sujeto \; a$$
$$15x_1 + 12x_2 + 7x_3 + 4x_4 \leq 33$$
$$x_2 = 0; \; x_3 = 0$$
$$x_1 \leq 2 \; x_2 \leq 2 \; x_3 \leq 3 \; x_4 \leq 2$$
$$x_i \geq 0 \text{ para } i = 1, 2, 3, 4$$

$\Rightarrow \; X_5 = (2, 0, 0, 0.75)$ y $z(X_5) = 39$

- **Problema** 6

$$max \; z = 18x_1 + 14x_2 + 8x_3 + 4x_4$$
$$sujeto \; a$$
$$15x_1 + 12x_2 + 7x_3 + 4x_4 \leq 33$$
$$x_2 = 0; \; x_3 = 1$$
$$x_1 \leq 2 \; x_2 \leq 2 \; x_3 \leq 3 \; x_4 \leq 2$$
$$x_i \geq 0 \text{ para } i = 1, 2, 3, 4$$

$\Rightarrow \; X_6 = (1.73, 0, 1, 0)$ y $z(X_6) = 39.2$

- **Problema** 7

$$max\ z = 18x_1 + 14x_2 + 8x_3 + 4x_4$$
$$sujeto\ a$$
$$15x_1 + 12x_2 + 7x_3 + 4x_4 \leq 33$$
$$x_2 = 0$$
$$x_3 = 2$$
$$x_1 \leq 2\ x_2 \leq 2\ x_3 \leq 3\ x_4 \leq 2$$
$$x_i \geq 0 \text{ para } i = 1, 2, 3, 4$$

$\Rightarrow$ $X_7 = (1.26, 0, 2, 0)$ y $z(X_7) = 38.8$

- **Problema** 8

$$max\ z = 18x_1 + 14x_2 + 8x_3 + 4x_4$$
$$sujeto\ a$$
$$15x_1 + 12x_2 + 7x_3 + 4x_4 \leq 33$$
$$x_2 = 0$$
$$x_3 = 3$$
$$x_1 \leq 2\ x_2 \leq 2\ x_3 \leq 3\ x_4 \leq 2$$
$$x_i \geq 0 \text{ para } i = 1, 2, 3, 4$$

$\Rightarrow$ $X_8 = (0.8, 0, 3, 0)$ y $z(X_8) = 38.4$

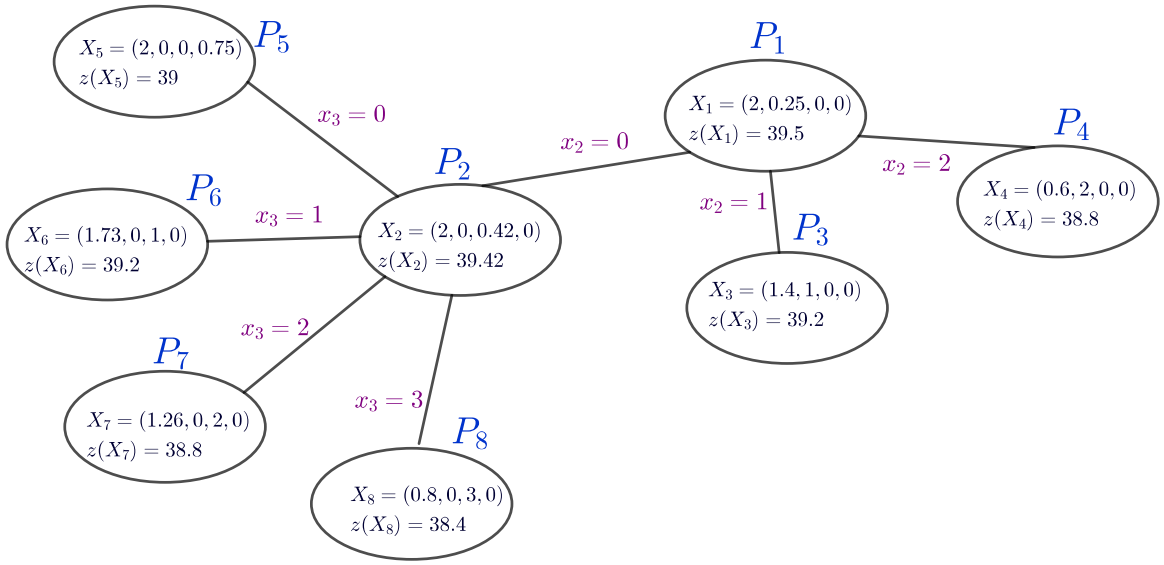


Figura 2.17: *Soluciones de los subproblemas* P_5, P_6, P_7 *y* P_8

Fuente: Elaboración propia utilizando Geogebra 5 (2016)

Iteración 3 Escogemos la mejor cota para seguir ramificando y existe un empate entre el problema 6 y el problema 3, escogemos el Problema 3 y se acotará a la variable x_1, las restricciones asociadas a ella son:

$$x_1 = 0\ para\ P_9$$
$$x_1 = 1\ para\ P_{10}$$
$$x_1 = 2\ para\ P_{11}$$

Los problemas a resolver son:

- **Problema** 9

$$max\ z = 18x_1 + 14x_2 + 8x_3 + 4x_4$$
$$sujeto\ a$$
$$15x_1 + 12x_2 + 7x_3 + 4x_4 \leq 33$$
$$x_2 = 1$$
$$x_1 = 0$$
$$x_1 \leq 2\ x_2 \leq 2\ x_3 \leq 3\ x_4 \leq 2$$
$$x_i \geq 0 \text{ para } i = 1, 2, 3, 4$$

$\Rightarrow\ X_9 = (0, 1, 3, 0)$
y $z(X_9) = 38$

- **Problema** 10

$$max\ z = 18x_1 + 14x_2 + 8x_3 + 4x_4$$
$$sujeto\ a$$
$$15x_1 + 12x_2 + 7x_3 + 4x_4 \leq 33$$
$$x_2 = 1$$
$$x_1 = 1$$
$$x_1 \leq 2\ x_2 \leq 2\ x_3 \leq 3\ x_4 \leq 2$$
$$x_i \geq 0 \text{ para } i = 1, 2, 3, 4$$

$\Rightarrow\ X_{10} = (1, 1, 0.85, 0)$
y $z(X_{10}) = 38.85$

- **Problema** 11

$$max\ z = 18x_1 + 14x_2 + 8x_3 + 4x_4$$
$$sujeto\ a$$
$$15x_1 + 12x_2 + 7x_3 + 4x_4 \leq 33$$
$$x_2 = 1$$
$$x_1 = 2$$
$$x_1 \leq 2\ x_2 \leq 2\ x_3 \leq 3\ x_4 \leq 2$$
$$x_i \geq 0 \text{ para } i = 1, 2, 3, 4$$

$\Rightarrow\ X_{11} = (2, 1, 0, 0)$
Este nodo se etiquetará como **Solución no factible**

En el problema 9 se encontró una solución entera, por lo que este nodo se etiquetará como solución candidata y los nodos 4, 7, 8, y 10 se etiquetarán como **nodo no prometedor** ya que el valor de z de estos es menor a 39 por lo que la mejor solución que se puede encontrar en los hijos es 38, esto se ilustra con el siguiente árbol:

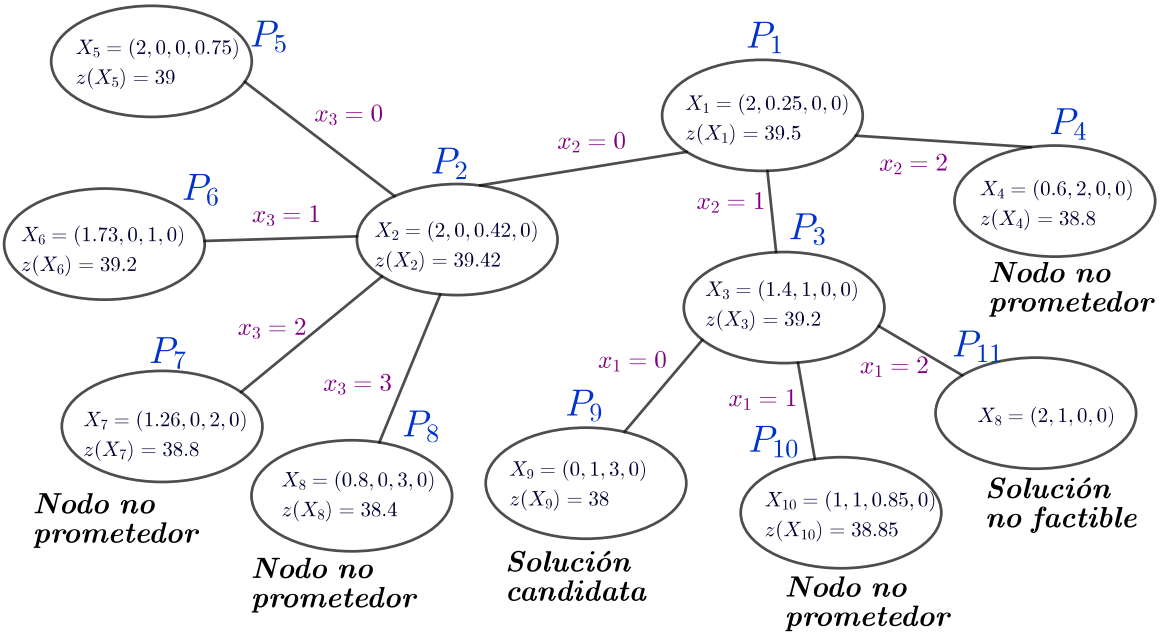


Figura 2.18: *Soluciones de los subproblemas* P_9,P_{10} *y* P_{11}

Fuente: Elaboración propia utilizando Geogebra 5 (2016)

Solo falta ramificar de los nodos 5 y 6, esto se desarrolla en las siguientes figuras:

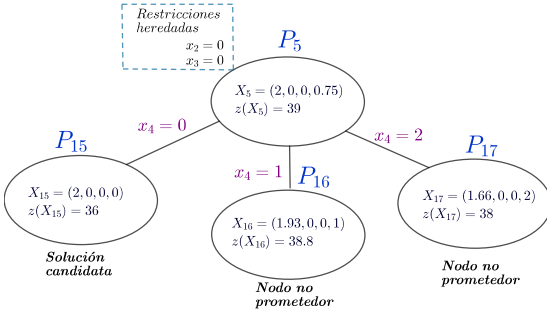


Figura 2.19: *Iteración 4*

Fuente: Elaboración propia utilizando Geogebra 5 (2016)

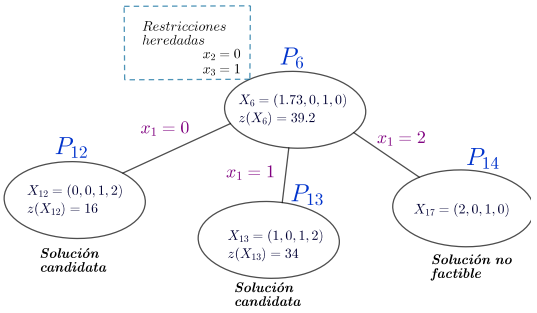


Figura 2.20: *Iteración 5*

Fuente: Elaboración propia utilizando Geogebra 5 (2016)

Como todos los nodos están etiquetados y existen problemas con etiqueta de "Solución candidata", obtenemos la mejor, esto es:

$$max\{36, 16, 34, 38\} = 38$$

Esta solución está asociada al problema 9, por lo que la solución óptima es:

$$X^* = (0, 1, 3, 0)$$

y

$$z(X^*) = 38$$

- **Problema binario**
  El problema relajado es:

$$\begin{gathered} max\ z = 50x_1 + 50x_2 + 64x_3 + 46x_4 + 50x_5 + 5x_6 \\ sujeto\ a \\ 56x_1 + 59x_2 + 80x_3 + 64x_4 + 75x_5 + 17x_6 \leq 190 \\ 0 \leq x_i \leq 1 \text{ para } i = 1, \ldots, 6 \end{gathered}$$

  cuya solución óptima es:

$$X_1 = (1, 1, 0.93, 0, 0, 0)$$

$$z(X_1) = 160$$

  En los problemas binarios solo tendremos dos ramas, si $x_i \notin Z$, las restricciones son:

$$x_i = 0$$

  y

$$x_i = 1$$

  Iteración 1 Como $x_3 \notin Z$ ramificamos con respecto a esta variable, las restricciones asociadas a x_3 son:

$$\begin{gathered} x_3 = 0 \\ x_3 = 1 \end{gathered}$$

  Quedando los problemas P_2 y P_3:

  - Problema 2

$$\begin{gathered} max\ z = 50x_1 + 50x_2 + 64x_3 + 46x_4 + 50x_5 + 5x_6 \\ sujeto\ a \\ 56x_1 + 59x_2 + 80x_3 + 64x_4 + 75x_5 + 17x_6 \leq 190 \\ x_3 = 0 \\ 0 \leq x_i \leq 1 \text{ para } i = 1, \ldots, 6 \\ \Rightarrow\ X_2 = (1, 1, 0, 1, 0.14, 0) \text{ y } z(X_2) = 153.33 \end{gathered}$$

  - Problema 3

$$\begin{gathered} max\ z = 50x_1 + 50x_2 + 64x_3 + 46x_4 + 50x_5 + 5x_6 \\ sujeto\ a \\ 56x_1 + 59x_2 + 80x_3 + 64x_4 + 75x_5 + 17x_6 \leq 190 \\ x_3 = 1 \\ 0 \leq x_i \leq 1 \text{ para } i = 1, \ldots, 6 \\ \Rightarrow\ X_3 = (1, 0.91, 1, 0, 0, 0) \text{ y } z(X_3) = 159.76 \end{gathered}$$

Al resolverlos obtenemos:

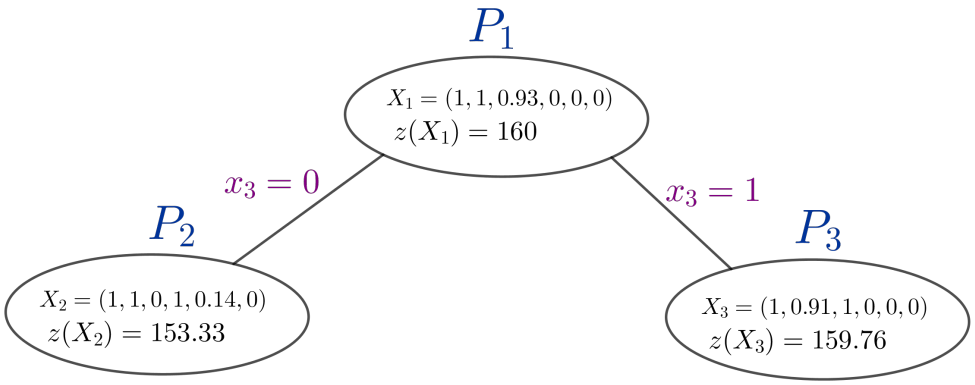


Figura 2.21: *Solución de los subproblemas P_2 y P_3*

Fuente: Elaboración propia utilizando Geogebra 5 (2016)

Como ninguna de las soluciones es entera, seguimos ramificando.

Iteración 2 Escogemos la mejor cota que es la asociada al problema 3, la variable que rompe la factibilidad es $x_2 \notin Z$ ramificamos con respecto a esta variable, las restricciones asociadas a x_2 son:

$$x_2 = 0$$
$$x_2 = 1$$

Quedando los problemas P_2 y P_3:

- Problema 4

$$max\ z = 50x_1 + 50x_2 + 64x_3 + 46x_4 + 50x_5 + 5x_6$$
$$sujeto\ a$$
$$56x_1 + 59x_2 + 80x_3 + 64x_4 + 75x_5 + 17x_6 \leq 190$$
$$x_3 = 1$$
$$x_2 = 0$$
$$0 \leq x_i \leq 1 \text{ para } i = 1, \ldots, 6$$
$$\Rightarrow\ X_4 = (1, 0, 1, 0.84, 0, 0) \text{ y } z(X_4) = 152.81$$

- Problema 5

$$max\ z = 50x_1 + 50x_2 + 64x_3 + 46x_4 + 50x_5 + 5x_6$$
$$sujeto\ a$$
$$56x_1 + 59x_2 + 80x_3 + 64x_4 + 75x_5 + 17x_6 \leq 190$$
$$x_3 = 1$$
$$x_2 = 1$$
$$0 \leq x_i \leq 1 \text{ para } i = 1, \ldots, 6$$
$$\Rightarrow\ X_5 = (0.35, 1, 1, 0, 0, 0) \text{ y } z(X_5) = 131.85$$

Al resolverlos obtenemos:

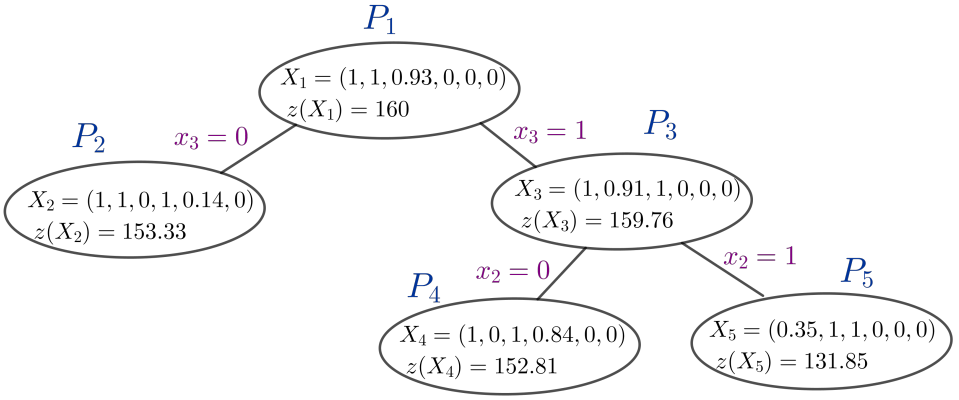


Figura 2.22: *Solución de los subproblemas P_4 y P_5*

Fuente: Elaboración propia utilizando Geogebra 5 (2016)

Seguimos iterando ya que no hemos encontrado alguna solución candidata.

Iteración 3 La mejor cota está asociada al problema 2, la variable que rompe la factibilidad es $x_5 \notin Z$, ramificamos con respecto a esta variable y las restricciones asociadas a x_5 son:

$$x_5 = 0$$
$$x_5 = 1$$

Quedando los problemas P_6 y P_7:

- Problema 6

$$max\ z = 50x_1 + 50x_2 + 64x_3 + 46x_4 + 50x_5 + 5x_6$$
$$sujeto\ a$$
$$56x_1 + 59x_2 + 80x_3 + 64x_4 + 75x_5 + 17x_6 \leq 190$$
$$x_3 = 0$$
$$x_5 = 0$$
$$0 \leq x_i \leq 1 \text{ para } i = 1, \ldots, 6$$
$$\Rightarrow\ X_6 = (1, 1, 0, 1, 0, 0.64) \text{ y } z(X_6) = 149.23$$

- Problema 7

$$max\ z = 50x_1 + 50x_2 + 64x_3 + 46x_4 + 50x_5 + 5x_6$$
$$sujeto\ a$$
$$56x_1 + 59x_2 + 80x_3 + 64x_4 + 75x_5 + 17x_6 \leq 190$$
$$x_3 = 0$$
$$x_5 = 1$$
$$0 \leq x_i \leq 1 \text{ para } i = 1, \ldots, 6$$
$$\Rightarrow\ X_7 = (1, 1, 0, 0, 1, 0) \text{ y } z(X_7) = 150$$

Al resolverlos obtenemos:

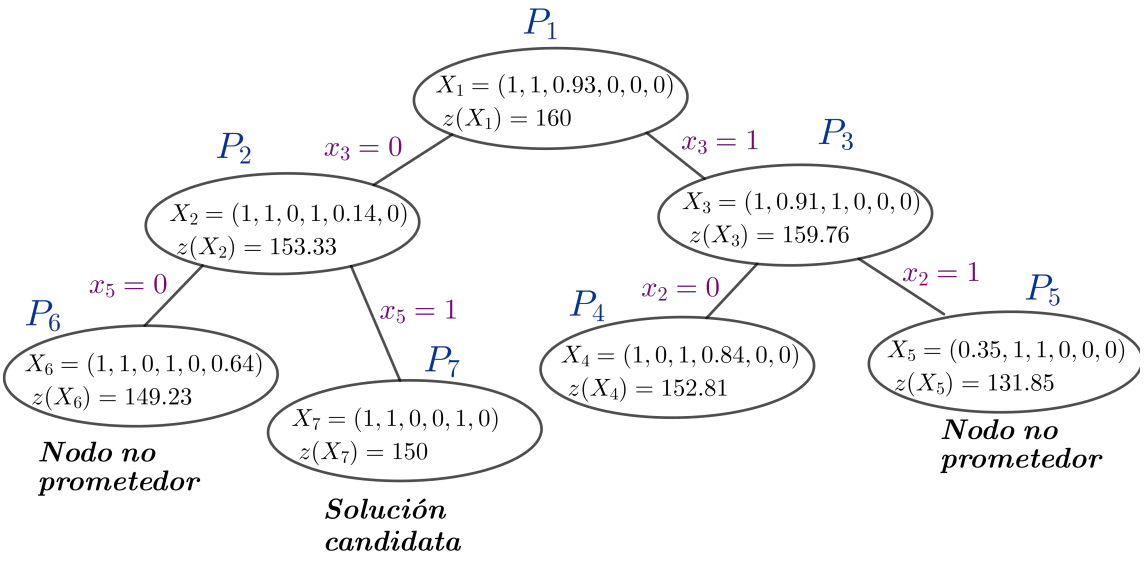


Figura 2.23: *Solución de los subproblemas P_6 y P_7*

Fuente: Elaboración propia utilizando Geogebra 5 (2016)

Observamos que tenemos una solución candidata en el problema 7 con $X_7 = (1, 1, 0, 0, 1, 0)$ y $z(X_7) = 150$, como los valores de la función objetivo en los problemas 5 y 6 son menores a 150 se etiquetarán como nodos no prometedores.

Como existe un nodo sin etiqueta, seguimos ramificando sobre el problema 4
Iteración 4 La variable que rompe la factibilidad es $x_4 \notin Z$, ramificamos con respecto a esta variable y las restricciones asociadas a x_4 son:

$$x_4 = 0$$
$$x_4 = 1$$

Quedando los problemas P_8 y P_9:

- Problema 8

$$max\ z = 50x_1 + 50x_2 + 64x_3 + 46x_4 + 50x_5 + 5x_6$$
$$sujeto\ a$$
$$56x_1 + 59x_2 + 80x_3 + 64x_4 + 75x_5 + 17x_6 \leq 190$$
$$x_3 = 1$$
$$x_2 = 0$$
$$x_4 = 0$$
$$0 \leq x_i \leq 1 \text{ para } i = 1, \ldots, 6$$
$$\Rightarrow\ X_8 = (1, 0, 1, 0, 0.72, 0) \text{ y } z(X_8) = 150$$

- Problema 9

$$max\ z = 50x_1 + 50x_2 + 64x_3 + 46x_4 + 50x_5 + 5x_6$$
$$sujeto\ a$$
$$56x_1 + 59x_2 + 80x_3 + 64x_4 + 75x_5 + 17x_6 \leq 190$$
$$x_3 = 1$$
$$x_2 = 0$$
$$x_4 = 1$$
$$0 \leq x_i \leq 1 \text{ para } i = 1, \ldots, 6$$
$$\Rightarrow\ X_9 = (0.82, 0, 1, 1, 0, 0) \text{ y } z(X_9) = 151.07$$

Al resolverlos obtenemos:

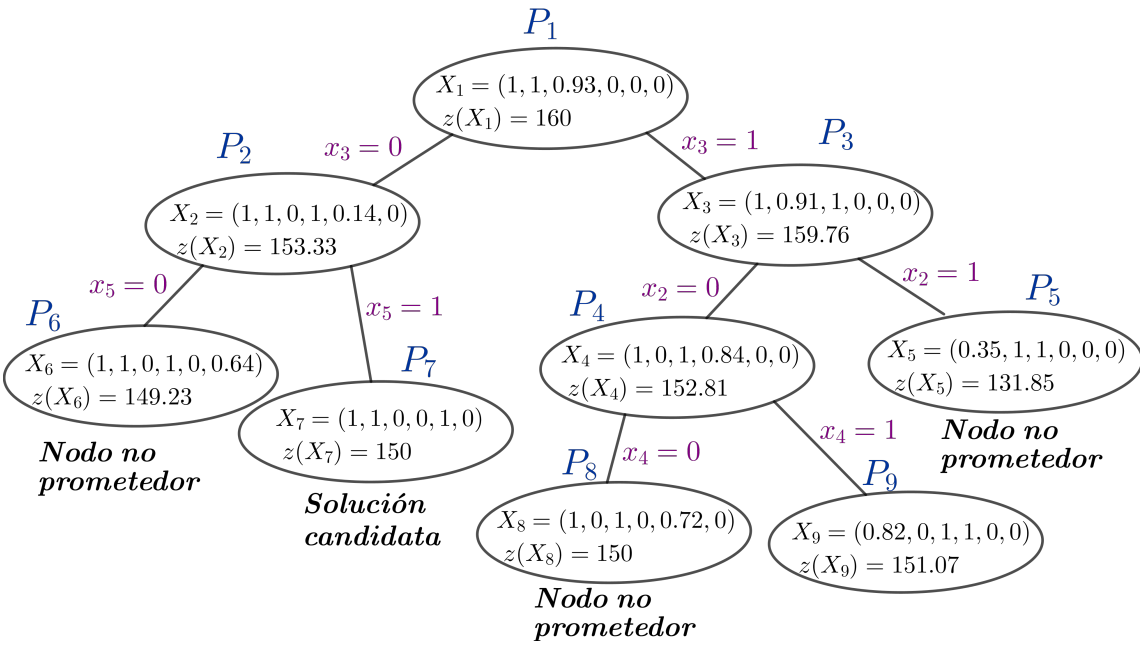


Figura 2.24: *Solución de los subproblemas P_8 y P_9*

Fuente: Elaboración propia utilizando Geogebra 5 (2016)

El valor de z obtenida en el problema 8 es 150 que es igual al del problema etiquetado como "Solución Candidata", por lo que se etiquetará como **nodo no prometedor** y seguiremos ramificando sobre el problema 9 cuya cota es mayor a 150

Iteración 5 La variable que rompe la factibilidad es $x_1 \notin Z$, ramificamos con respecto a esta variable y las restricciones asociadas a x_1 son:

$$x_1 = 0$$
$$x_1 = 1$$

Quedando los problemas P_8 y P_9:

- Problema 10

$$max\ z = 50x_1 + 50x_2 + 64x_3 + 46x_4 + 50x_5 + 5x_6$$
$$sujeto\ a$$
$$56x_1 + 59x_2 + 80x_3 + 64x_4 + 75x_5 + 17x_6 \leq 190$$
$$x_3 = 1$$
$$x_2 = 0$$
$$x_4 = 1$$
$$x_1 = 0$$
$$0 \leq x_i \leq 1 \text{ para } i = 1, \ldots, 6$$
$$\Rightarrow\ X_{10} = (0, 0, 1, 1, 0.61, 0) \text{ y } z(X_{10}) = 146.66$$

- Problema 11

$$max\ z = 50x_1 + 50x_2 + 64x_3 + 46x_4 + 50x_5 + 5x_6$$
$$sujeto\ a$$
$$56x_1 + 59x_2 + 80x_3 + 64x_4 + 75x_5 + 17x_6 \leq 190$$
$$x_3 = 1$$
$$x_2 = 0$$
$$x_4 = 1$$
$$x_1 = 1$$
$$0 \leq x_i \leq 1 \text{ para } i = 1, \ldots, 6$$
$$\Rightarrow\ X_{11} = (1, 0, 1, 1, 0, 0)$$
Que es una solución no factible.

Al resolverlos obtenemos:

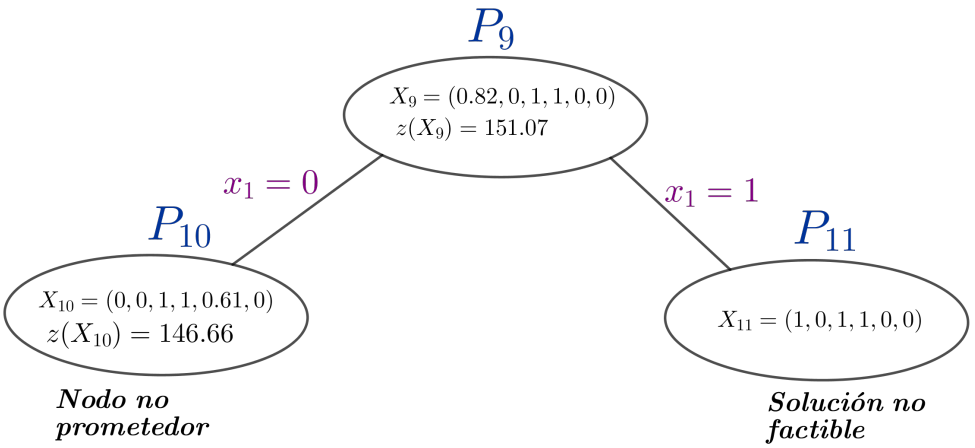


Figura 2.25: *Solución de los subproblemas P_{10} y P_{11}*

Fuente: Elaboración propia utilizando Geogebra 5 (2016)

Todos los nodos han sido etiquetados, entonces, obtenemos la solución óptima, observamos que solo existe un nodo con la etiqueta de **solución candidata**, por lo que ésta es la solución óptima para el problema entero y es la solución asociada al nodo 7

$$X^* = (1, 1, 0, 0, 1, 0)$$

y

$$z(X^*) = 150$$

Después de resolver tres problema de la mochila con ramificación y acotamiento podemos concluir que el número de problemas que se deben resolver para llegar a la solución óptima es incierto ya que cada problema se comporta diferente; analicemos el problema binario:

$$max\ z = 50x_1 + 50x_2 + 64x_3 + 46x_4 + 50x_5 + 5x_6$$
$$sujeto\ a$$
$$56x_1 + 59x_2 + 80x_3 + 64x_4 + 75x_5 + 17x_6 \leq 190$$
$$0 \leq x_i \leq 1 \text{ para } i = 1, \ldots, 6$$
$$x_i \in Z \text{ para } i = 1, \ldots, 6$$

Cuadro 2.4: *Alternativas totales y subproblemas analizados en el problema de la mochila binario*

Número de artículos	Número de alternativas	Número de problemas analizados con ramificación y acotamiento
6	64	11

A pesar de que puede crecer exponencialmente el número de subproblemas que se deben analizar con ramificación y acotamiento, nunca vamos a analizar todas las alternativas ya que tenemos ciertos criterios (etiquetas) que reducen dichos problemas, como: "Nodo no prometedor" y "Solución no factible"
Pero si podemos hacer la observación: Si tenemos una instancia con más artículos crecerá el número de subproblema a analizar, por esta razón el problema de la mochila se vuelve intratable con ramificación y acotamiento (para instancias grandes.)

2.2. Programación dinámica

La programación dinámica es utilizada para solucionar problemas de optimización y está basada en la partición de los últimos en una serie de etapas. Es decir, en una serie de subproblemas ligados entre sí, éstos son más sencillos de resolver ya que sólo se requiere de una decisión por lo que al utilizar programación dinámica se podría decir que se está utilizando un proceso de decisión en multietapas.
Los subproblemas se definen de tal manera que la respuesta global está constituida por una combinación de las decisiones tomadas en cada una de las etapas.
La programación dinámica está basada en la teoría que desarrolló Richard Bellman a principios de 1950 [Bertsekas (1987)], la cual incluye el *Principio de Optimalidad* que dice:

Dado el estado actual del sistema, la decisión óptima para cada una de las etapas restantes no debe depender de estados previamente alcanzados o de decisiones previamente tomadas

Es decir, que las decisiones que se toman en cada una de las etapas son independientes de las que se hayan tomado en etapas anteriores o de estados previamente alcanzados.
Para plantear un problema como uno de programación dinámica es necesario empezar con las variables de éste, se pueden identificar dos tipos: [Hernández A, 1994]

- **Variables de estado:** Son la liga entre las etapas y se podría decir que son una medida de las condiciones que existen al empezar alguna etapa, nos ayudan a relacionar las soluciones (decisiones) de cada una da las etapas anteriores con la etapa actual. Regularmente las variables de estado se representan con Y_j donde j es la etapa actual.

- **Variables de decisión:** Son las variables que involucran las alternativas posibles en cada una de las etapas del problema de optimización. Las variables de decisión se representan con x_i.

2.2.1. Elementos de un problema de programación dinámica

En un problema de optimización visto con el enfoque de la programación dinámica se identifican cuatro elementos fundamentales: etapas, alternativas, estados del sistema y ecuaciones recursivas.

- *Etapa:* Es un subproblema del problema de optimización, en el cual se debe de tomar una decisión.
  Es importante conocer bien la estructura del problema para poder dividir al problema en etapas (en subproblemas)

- *Alternativas:* Se representan mediante las variables de decisión de cada etapa j (x_j), como su nombre lo dice, son todos los posibles valores que puede tomar (x_j), éstas influyen directamente en la función de rendimiento del problema.

- *Estados del sistema:* Son la liga o la relación que existe entre las etapas. El estado del sistema debe contener toda la información necesaria para poder tomar una decisión factible en la etapa actual. Cabe señalar que la definición de estado deber permitir que se tome una decisión factible para la etapa actual sin necesidad de comprobar las decisiones realizadas en etapas anteriores.
  Los estado del sistema se representan con las variables de estado Y_j

- *Ecuaciones recursivas:* Contienen toda la información del rendimiento acumulado durante las etapas anteriores, de tal manera que la ecuación recursiva de la etapa i contiene toda la información necesaria para tomar una decisión en la etapa $i+1$, por lo que en la última etapa se tiene el rendimiento óptimo total (o viceversa: la ecuación de la etapa i contiene toda la información necesaria para tomar una decisión en la etapa $i-1$, por lo que en la primera etapa se tiene el rendimiento óptimo total, esto es porque en la programación dinámica se puede resolver un problema empezando con la etapa 1 hasta la etapa n o empezando con la etapa n y terminando con la etapa 1).

Cada etapa tiene una variable de estado diferente, esto nos hace pensar que debemos tener una función que nos proporcione una transformación de estados, sea f_i dicha transformación, pero ¿quiénes influyen en esta transformación?. En cada etapa podemos observar lo siguiente: [Prawda (1976)]

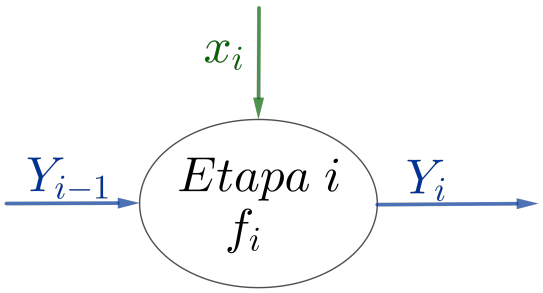


Figura 2.26: *Transformación de estados*

Fuente: Elaboración propia utilizando Geogebra 5 (2016)

En cada etapa i tenemos que tomar una decisión x_i que va a estar ligada con un estado inicial Y_{i-1} y al terminar la etapa tendremos un estado final Y_i, por lo que podemos definir a f_i en función del estado inicial Y_{i-1} y la decisión tomada en la etapa x_i, esto es:

$$Y_i = f_i(Y_{i-1}, x_i)$$

Esto se repite en cada una de las etapas:

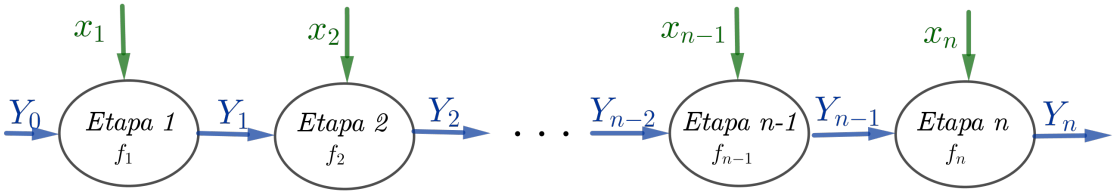


Figura 2.27: *Variables por etapas*

Fuente: Elaboración propia utilizando Geogebra 5 (2016)

Por lo que

$$Y_n = f_n(Y_{n-1}, x_n)$$

pero

$$Y_{n-1} = f_{n-1}(Y_{n-2}, x_{n-1})$$

Con esto podemos concluir que :

$$Y_n = f_n(f_{n-1}(\ldots(f_2(f_1(Y_0, x_1), x_2), \ldots), x_{n-1}), x_n)$$

Por lo que Y_n, el estado en la etapa n; depende únicamente del vector de entrada Y_0 y el conjunto de decisiones parciales de cada una de las etapas $(x_n, x_{n-1}, \ldots, x_2, x_1)$. Sucede algo similar con la función de rendimiento g_i, si denotamos con B_i el beneficio obtenido en la etapa i, entonces g_i está en función de la variable de estado Y_i y la decisión tomada en la etapa i

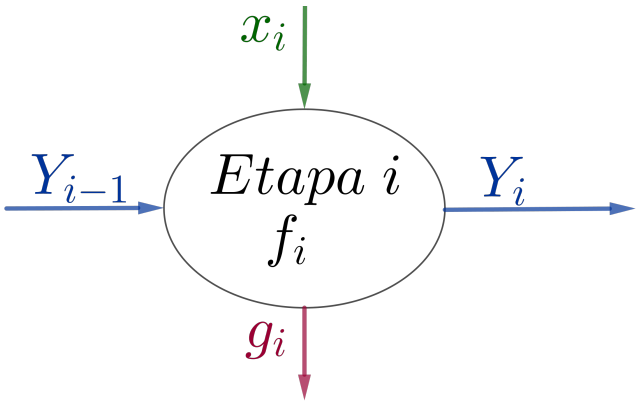


Figura 2.28: *Función de rendimiento por etapas*

Fuente: Elaboración propia utilizando Geogebra 5 (2016)

Esto también se repite etapa con etapa como se muestra en el siguiente esquema:

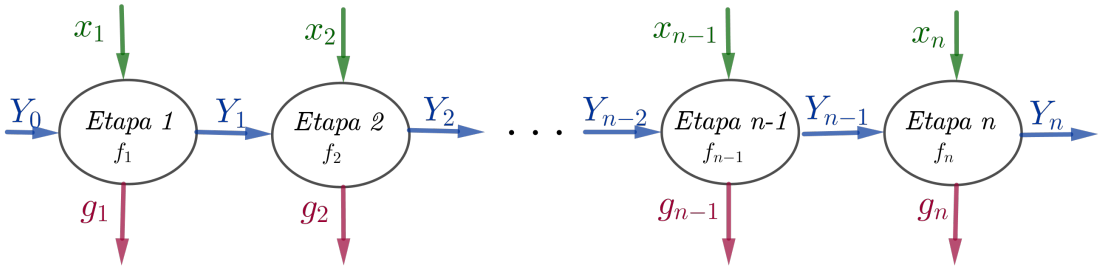


Figura 2.29: *Etapas de un problema de programación dinámica*

Fuente: Elaboración propia utilizando Geogebra 5 (2016)

La función de rendimiento es:

$$B_1 = g_1(Y_1, x_1)$$
$$B_2 = g_1(Y_2, x_2)$$
$$\vdots$$
$$B_{n-1} = g_{n-1}(Y_{n-1}, x_{n-1})$$
$$B_n = g_n(Y_n, x_n)$$

Ecuación Recursiva

La ecuación recursiva guarda el beneficio acumulado (o costo acumulado). Como se desea maximizar el beneficio (o minimizar el costo), se debe maximizar (o minimizar) la función:

$$G_n = max_{\{x_n, x_{n-1}, ..., x_2, x_1\}}[g_n(Y_n, x_n) + g_{n-1}(Y_{n-1}, x_{n-1}) + \ldots + g_1(Y_1, x_1)]$$
$$sujeto\ a$$
$$Y_i = f_i(Y_{i-1}, x_i)\ para\ i = n, n-1, \ldots, 2, 1$$

Observación 9 *Para cualquier par de funciones reales $p(x)$ y $q(x, y)$*

$$max_{x,y}[p(x) + q(x, y)] = max_x[p(x) + max_y q(x, y)]$$

Si lo anterior lo aplicamos a G_n obtenemos lo siguiente:

$$G_n = max_{x_n}\{g_n(Y_n, x_n) + max_{x_{n-1}, ..., x_2, x_1}[g_{n-1}(Y_{n-1}, x_{n-1}) + ... + g_1(Y_1, x_1)]\}$$
$$sujeto\ a$$
$$Y_i = f(Y_{i-1}, x_i)\ para\ i = 1, 2, ..., n-1, n.$$

Si hacemos

$$G_{n-1} = max_{x_{n-1}, ..., x_2, x_1}[g_{n-1}(Y_{n-1}, x_{n-1}) + ... + g_1(Y_1, x_1)]$$

Sustituimos en G_n y obtenemos

$$G_n(Y_n) = max_{x_n}[g_n(Y_n, x_n) + G_{n-1}]$$

Por lo que la ecuación recursiva en forma general es:

$$G_n(Y_n) = max_{x_n}[g_n(Y_n, x_n) + G_{n-1}]$$
$$sujeto\ a$$
$$Y_i = f(Y_{i-1}, x_i)\ i = 1, 2, ..., n-1, n.$$

Una ventaja de resolver un problema de optimización con programación dinámica es que se puede resolver el problema de la etapa 1 hasta la etapa n o bien desde la etapa n hasta la etapa 1; esto es posible ya que la decisión que se toma en cada etapa es independiente a la decisión de la o las etapa(s) anteriores y todos los esquemas anteriores bien pueden empezar por la etapa n y finalizar en la etapa 1 sin ningún problema.

2.2.2. Programación dinámica aplicada al problema de la ruta más corta

El problema de la ruta más corta es muy estudiado por las aplicaciones que tiene y los diversos métodos que existen para resolverlo [Ahuja (1988)], la programación dinámica es uno de ellos.
Para poder entender los conceptos necesarios para plantear el problema de la ruta más corta utilizando programación dinámica utilizaremos una instancia del problema.

El problema se plantea a partir de la siguiente situación:
Una persona desea salir de la ciudad 1 (C_1) y llegar a la ciudad 7 (C_7) para esto es posible pasar por ciudades intermedias, los datos que se conocen son:

- Las conexiones entre ciudades.
- Las distancias entre las ciudades.

Se desea llevar a cabo dicho recorrido de tal manera que sea a una distancia mínima. En la siguiente gráfica cada nodo representa una ciudad y cada flecha representa un camino que une a la ciudad i con la ciudad j, el número asociado a una flecha representa la distancia que existe entre dichas ciudades

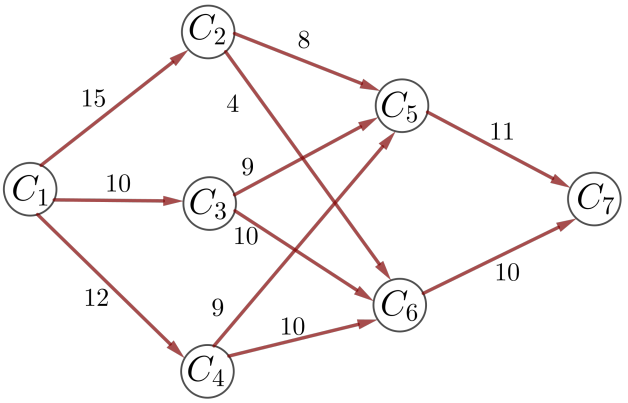


Figura 2.30: *Representación del problema de la ruta más corta mediante una gráfica*

Fuente: Elaboración propia utilizando Geogebra 5 (2016)

Si la variable se decisión se define como:

$$x_{ij} = \begin{cases} 1 \quad si \quad se\ recorre\ el\ camino\ (i,j) \\ \\ 0 \quad si \quad no\ se\ recorre\ el\ camino\ (i,j) \end{cases}$$

El problema de la ruta más corta planteado como uno de programación lineal entera es:

$$Min\ z = 15x_{12}+10x_{13}+12x_{14}+8x_{25}+4x_{26}+9x_{35}+10x_{36}+9x_{45}+10x_{46}+11x_{57}+10x_{67}$$

$$sujeto\ a$$

$$x_{12} + x_{13} + x_{14} = 1$$

$$x_{25} + x_{26} - x_{12} = 0$$

$$x_{35} + x_{36} - x_{13} = 0$$

$$x_{45} + x_{46} - x_{14} = 0$$

$$x_{57} - x_{25} - x_{35} - x_{45} = 0$$

$$x_{67} - x_{26} - x_{36} - x_{46} = 0$$

$$-x_{57} - x_{67} = -1$$

$$x_{ij} \in \{0, 1\}\ \forall (i, j) \in A(G)$$

Resuelto con Lingo obtenemos el camino:

$$C_1 \Rightarrow C_2 \Rightarrow C_6 \Rightarrow C_7$$

Con una distancia

29

Ahora vamos a plantearlo con la metodología de la programación dinámica tomando en cuenta que lo resolveremos empezando en la etapa n
Las variables del problema son:

- **Variables de estado:** Nos indica lo siguiente: si estamos situados en la ciudad j ¿De qué ciudad venimos?, a esto se le conoce como predecesor y se denota $\Gamma^{-}(j)$

- **Variables de decisión:** Éstas nos indican todas las alternativas que podemos alcanzar en cada etapa, es decir, son todas las ciudades contenidas en la etapa j.

Los elementos los podemos definir:

- **Etapa:** Es un bloque alcanzado, esto se explica con la siguiente figura:

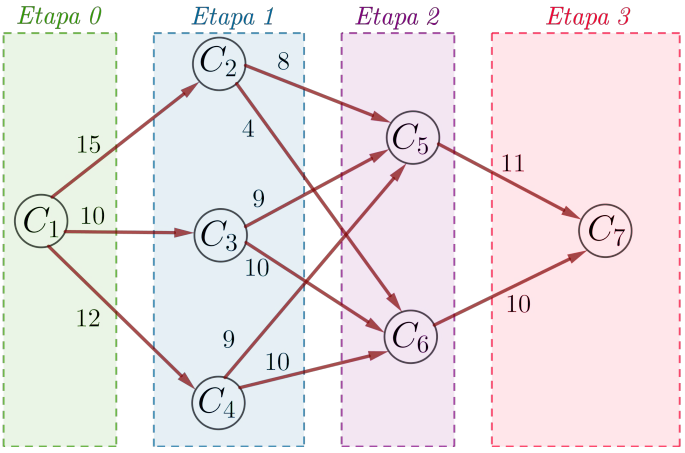


Figura 2.31: *Etapas del problema de la ruta más corta*

Fuente: Elaboración propia utilizando Geogebra 5 (2016)

- **Alternativas:** Nos indican las ciudades a las que podemos llegar en cada etapa, por ejemplo en la etapa 2, las ciudades que podemos alcanzar son:
  - Ir a C_5
  - Ir a C_6
- **Estados del sistema:** Como en el problema de la ruta más corta se desea ir formando una ruta, el estado del sistema en la etapa actual nos indica el predecesor de la alternativa x_i, es decir, si estamos en la etapa k, los estados del sistema son: $\Gamma^-(i)\ \forall i \in \{etapa\ K\}$
  Por ejemplo si estamos en la estapa 2, los estados del sistema son: C_2, C_3 *y* C_4
  Para ligar la solución con la solución de la etapa $K+1$, la transformación de estados es:

$$Y_{i+1} = \Gamma^+(x_i)$$

- **Ecuaciones recursivas:** El rendimiento en el problema de la ruta más corta está en función de la distancia recorrida, por lo que si la función de rendimiento de la etapa i la denotamos por d_i la cual depende de la variable de estado Y_i y de x_i, es decir, depende del predecesor y de la decisión que se tomo en esa etapa la podemos expresar:

$$g_i(Y_i, x_i) = d(Y_i, x_i)$$

Como el objetivo del problema de la ruta más corta es minimizar la distancia recorrida, la ecuación recursiva es:

$$G_1 = min_{x_1,x_2,\ldots x_n} = [d_1(Y_1, x_1) + d_2(Y_2, x_2) + \ldots + d_n(Y_n, x_n)]$$

La transformación de estados es:

$$Y_{i+1} = \Gamma^{+}(x_i)$$

Si hacemos

$$G_2(Y_2) = min_{x_2,\ldots,x_{n-1},x_n}[d_2(Y_2, x_2) + \ldots + d_n(Y_n, x_n)]$$

Obtenemos:

$$G_1 = min_{x_1} = [d_1(Y_1, x_1) + G_2(Y_2)]$$

$$sujeto\ a$$

$$Y_2 = \Gamma^{+}(x_i)$$

Si repetimos esto en cada etapa, podemos expresar la ecuación recursiva para cualquier etapa j

$$G_j(Y_j) = min_{x_j}[d_j(Y_j, x_j) + G_{j+1}(Y_{j+1})]$$

$$sujeto\ a$$

$$Y_{j+1} = \Gamma^{+}(x_j)$$

Toda la información de cada etapa estará contenida en la siguientes tabla:

Cuadro 2.5: *Información del problema por etapa*

Variable de estado Y_i Predecesor de las ciudades que se pueden alcanzar en la etapa i	Variable de decisión x_i Ciudades que se pueden alcanzar en la etapa i	G_i^*=$min\{G_i\}$ Mejor valor de la función de rendimiento asociada a la variable de estado Y_i	x_i^* óptima Decisión asociada a G_i^*
	Alt 1 | Alt 2 |...| Alt r		
Predecesor *(Pred)*	*Ecuación recursiva* $G_n(Y_n) = min_{x_n}[d_n(Y_n, x_n) + G_{n-1}]$		

Recordemos el problema dividido por etapas:

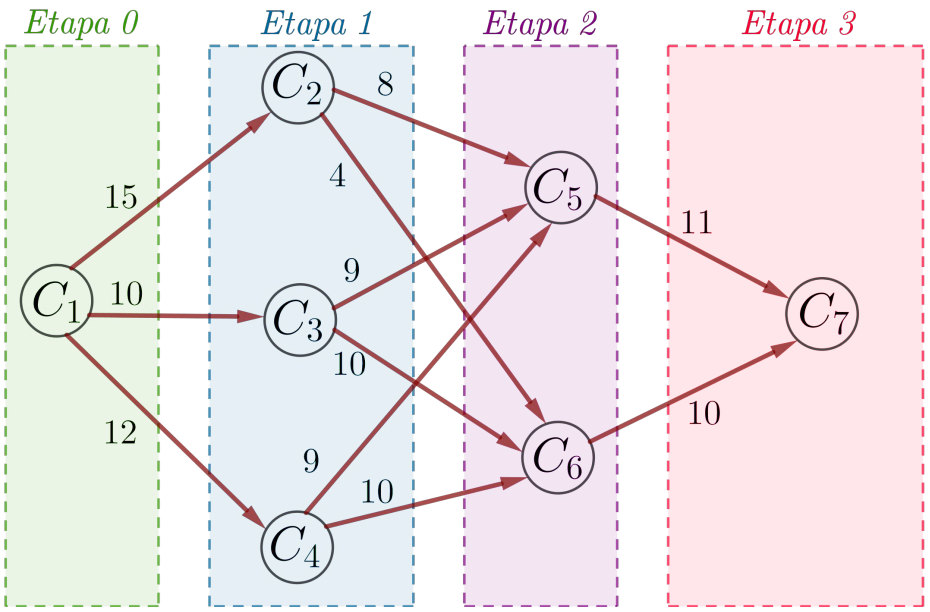


Figura 2.32: *Etapas del problema*

Fuente: Elaboración propia utilizando Geogebra 5 (2016)

Iniciamos en la etapa 3.

Etapa 3: $G_4(Y_4) = 0$ A esto se le conoce como condición frontera, para poder utilizar las ecuaciones recursivas.

En esta etapa la única alternativa es C_7 por lo que x_3 solo puede tener el valor C_7

Los estados son los $\Gamma^-(C_7) = \{C_5, C_6\} \Rightarrow Y_3 = C_5$ *y* $Y_3 = C_6$

Calculemos la función de rendimiento utilizando la ecuación recursiva:

$$G_3(Y_3) = [d_3(Y_3, x_3) + G_4(Y_4)]$$

$$sujeto\ a$$

$$Y_4 = \Gamma^+(x_3)$$

Sustituyendo, obtenemos:

Para $Y_3 = C_5$

$$G_3(C_5) = d(C_5, C_7) + G_4(Y_4) = 11 + 0 = 11$$

Para $Y_3 = C_6$

$$G_3(C_6) = d(C_6, C_7) + G_4(Y_4) = 10 + 0 = 10$$

Vaciamos esta información en la tabla

Cuadro 2.6: *Etapa* 3 *Ruta más corta*

Y_3	C_7	$G_3^*(Y_3) = min\{G_3(Y_3)\}$	x^*
C_5	11	11	C_7
C_6	10	10	C_7

Etapa 2: En esta etapa las alternativas son C_5 y C_6 por lo que x_2 puede tener esos valores.
Los estados son los $\Gamma^-(C_5) \cup \Gamma^-(C_6) = \{C_2, C_3, C_4\} \Rightarrow Y_2 = C_2,\ Y_2 = C_3$ *y* $Y_2 = C_4$
La ecuación recursiva para esta etapa es:

$$G_2(Y_2) = [d_2(Y_2, x_2) + G_3(Y_3)]$$

$$sujeto\ a$$

$$Y_3 = \Gamma^+(x_2)$$

Sustituyendo, obtenemos:
Para $Y_2 = C_2$

$$G_2(C_2) = d(C_2, C_5) + G_3(C_5) = 8 + 11 = 19$$
$$G_2(C_2) = d(C_2, C_6) + G_3(C_6) = 4 + 10 = 14$$

Para $Y_2 = C_3$

$$G_2(C_3) = d(C_3, C_5) + G_3(C_5) = 9 + 11 = 20$$
$$G_2(C_3) = d(C_3, C_6) + G_3(C_6) = 10 + 10 = 20$$

Para $Y_2 = C_4$

$$G_2(C_4) = d(C_4, C_5) + G_3(C_5) = 9 + 11 = 20$$
$$G_2(C_4) = d(C_4, C_6) + G_3(C_6) = 10 + 10 = 20$$

Vaciamos esta información en la tabla

Cuadro 2.7: *Etapa* 2 *Ruta más corta*

Y_2	C_5	C_6	$G_2^*(Y_2) = min\{G_2(Y_2)\}$	x^*
C_2	$8 + 11 = 19$	$4 + 10 = 14$	$min\{19, 14\} = 14$	C_6
C_3	$9 + 11 = 20$	$10 + 10 = 20$	$min\{20, 20\} = 20$	C_5, C_6
C_4	$9 + 11 = 20$	$10 + 10 = 20$	$min\{20, 20\} = 20$	C_5, C_6

Etapa 1: En esta etapa las alternativas son C_2, C_3 y C_4 por lo que x_1 puede tener esos valores.
Los estados son los $\Gamma^-(C_2) \cup \Gamma^-(C_3) \cup \Gamma^-(C_4) = \{C_1\} \Rightarrow Y_1 = C_1$
La ecuación recursiva para esta etapa es:

$$G_1(Y_1) = [d_1(Y_1, x_1) + G_2(Y_2)]$$

$$sujeto\ a$$

$$Y_2 = \Gamma^+(x_1)$$

Sustituyendo, obtenemos:
Para $Y_1 = C_1$

$$G_1(C_1) = d(C_1, C_2) + G_2(C_2) = 15 + 14 = 29$$
$$G_1(C_1) = d(C_1, C_3) + G_2(C_3) = 10 + 20 = 30$$
$$G_1(C_1) = d(C_1, C_4) + G_2(C_4) = 12 + 20 = 32$$

Vaciamos esta información en la tabla

Cuadro 2.8: *Etapa* 1 *Ruta más corta*

Y_1	C_2	C_3	C_4	$G_1^*(Y_1) = min\{G_1(Y_1)\}$	x^*
C_1	$15 + 14 = 29$	$10 + 20 = 30$	$12 + 20 = 32$	$min\{29, 30, 32\} = 29$	C_2

Ahora recuperemos la ruta: Iniciamos en la etapa 1, la mejor opción para C_1 es C_2
Pasamos a la etapa 2, la mejor opción para C_2 es C_6
Por último vamos a la etapa 3 y la mejor opción para C_6 es C_7.
Por lo tanto la ruta es:

$$C_1 \Rightarrow C_2 \Rightarrow C_6 \Rightarrow C_7$$

Con una distancia 29

Que coincide con la obtenida con el modelo lineal resuelto con Lingo

2.2.3. Programación dinámica aplicada al problema de la mochila

Recordemos el modelo matemático del problema general de la mochila:

$$max\ z = \sum_{i=1}^{n} c_i x_i$$
$$sujeto\ a\ \sum_{i=1}^{n} a_i x_i \leq b$$
$$x_i \geq 0\ y\ x_i \in Z \text{ para } i = 1, ..., n.$$

Donde:

- c_i es el beneficio obtenido por unidad del artículo i, $i = 1, 2, ..., n$.
- x_i es el número de unidades incluidas del artículo i, $i = 1, 2, ..., n$.
- a_i es el peso del artículo i, $i = 1, 2, ..., n$.
- b es la capacidad de la mochila.

Por lo que los elementos del problema de programación dinámica enunciados anteriormente en el caso del problema de la mochila son los siguientes:

- **Etapa:** En el problema de la mochila cada artículo constituye una etapa, es decir, el subproblema i es determinar el número de unidades del artículo i que deben incluirse para que se maximice el beneficio en esa etapa.

- **Alternativas:** El número de alternativas en cada etapa es $\left[\frac{b}{a_i}\right]+1$, ya que son los posibles valores que puede tomar x_i en la etapa i, es decir, $x_i = 0, 1, ..., \left[\frac{b}{a_i}\right]$.

- **Estados del sistema:** Y_i, la variable de estado, representa la capacidad disponible de la mochila en cada etapa y por tanto la ecuación de transformación de estados es:
$$Y_{i-1} = Y_i - (a_i * x_i).$$

- **Ecuaciones recursivas:** La ecuación recursiva general es
$$\begin{array}{l} G_n(Y_i) = \text{máx}_{x_i}[g_i(Y_i, x_i) + G_{i-1}(Y_{i-1})] \\ \textit{sujeto a} \\ Y_{i-1} = Y_i - (a_i * x_i) \quad i = 1, 2, ..., n-1, n. \\ \qquad\qquad G_0 = 0 \qquad \text{y} \qquad Y_n = b. \end{array}$$
Donde la función con la cual obtenemos el beneficio en cada iteración de cada etapa es $g_i(y_i, x_i) = a_i * x_i$ por lo que la ecuación recursiva es:
$$\begin{array}{l} G_i(Y_i) = \text{máx}_{x_i}[c_i * x_i + G_{i-1}(Y_i - (a_i * x_i))] \\ \textit{sujeto a} \\ Y_{i-1} = Y_i - (a_i * x_i) \quad i = n, n-1, ...2, 1. \end{array}$$

2.2.4. Algoritmo

Datos del problema
n es el número de elementos (artículos).

c es el vector de utilidades: $(c_1, c_2, ..., c_n)$.

a es el vector de pesos: $(a_1, a_2, ..., a_n)$.

b es la capacidad de la mochila.
Inicialización $q = 1$ (q es la variable que representará el número de etapa en la cual se está iterando).
Paso 1) Si $m = \left[\frac{b}{a_q}\right]$, entonces $m+1$ es el número de alternativas de la etapa q, ya que estas son $x_q = 0, 1, \ldots, m$.

Por cada etapa se hace la siguiente tabla:

Cuadro 2.9: *Cuadro con las información de cada iteración*

Fuente: Elaboración propia utilizando PowerPoint (2010)

Alt. / Y_{qi}	$x_q = 0$	$x_q = 1$	. . .	$x_q = b/\ a_q$	$G_q = (Y_{qi})$	Alt. Óptima
0						
1						
.						
.						
.						
b						

Paso 2) *Llenado de tabla.*
Llenado de las columnas $x_{q's}$.

- Si la alternativa no es factible, es decir, que $x_q * a_q > Y_{qi}$, entonces se coloca '__'

- Si la alternativa es factible, es decir, que $x_q * a_q \leq Y_{qi}$, se coloca $x_q * c_q + g_{q-1}(Y_{qi} - x_q * a_q)$ con la condición inicial $G_0 = 0$ (condición frontera.

Llenado de la columna $G_q(Y_{qi})$.
Se obtiene el valor máximo por fila, es decir, $G_q(Y_{qi})$ es el máximo beneficio por cada valor de Y_{qi}.
Llenado de la columna alternativa óptima.
A esta columna le corresponde el valor de la variable de decisión para la cual obtenemos el beneficio máximo.

- $q = q + 1$

¿Es $q < n$? Si sí ir al paso 1, si no ir al paso 3.
Paso 3) Recuperación de la solución óptima.
Observación: El valor de b (capacidad de la mochila) puede cambiar, pero sólo puede ser menor o igual al que se dio inicialmente.
Sea:

$$q = n$$

$$Y_q = b$$

1. Se toma la alternativa óptima de la etapa q para el valor Y_q.
2. Se realiza la transformación de estados:

$$Y_{q-1} = Y_q - (a_i * x_i)$$

3. Actualizamos $q = q - 1$
4. ¿Si $q > 1$? ir al paso 1, si no terminar, ya se tiene la solución óptima.

El beneficio máximo es el que se encuentra en la última etapa, en la columna $G_n(Y_{nb})$ y en la fila $Y_n = b$.

Ejemplo 9 *Ahora resolveremos una instancia del problema de la mochila general utilizando programación dinámica.*
Consideremos el siguiente problema:

$$\begin{gathered} max\ z = 6x_1 + 16x_2 + 10x_3 + 20x_4 \\ sujeto\ a \\ 6x_1 + 14x_2 + 9x_3 + 17x_4 \leq 40 \\ x_i \geq 0\ para\ i = 1, 2, 3, 4 \end{gathered}$$

Este problema se resolverá empezando en la etapa 1
Los elementos del problema son:

- Por tener 4 artículos se desarrollará en 4 etapas.
- Los estados del sistema se asocian a la capacidad de la mochila, entonces, van desde 0 hasta 40.
- Las alternativas son diferentes para cada etapa, se debe realizar el cociente $\left[\frac{b}{a_i}\right]$
- Las ecuaciones recursivas guardan el beneficio por etapa ligado con el beneficio de las etapas anteriores.

Etapa 1
Los datos para esta etapa son:

- $c_1 = 6$
- $a_1 = 6$
- $b = 40$

El número de alternativas es:

$$\left[\frac{b}{a_1}\right] = \left[\frac{40}{6}\right] = [6.666] = 6$$

La ecuación recursiva que se utilizará en esta etapa es:

$$G_1(Y_1) = max_{x_1}\{c_1x_1 + G_0(Y_0)\}$$

$$sujeto\ a$$

$$0 \leq Y_1 \leq 40$$

$$x_1 = 0, 1, \ldots \left[\frac{b}{a_1}\right]$$

Tomando en cuenta que $G_0(Y_0) = 0$ (condición frontera)

Llenado de la tabla

En la primer columna solo se anotan los valores de Y_1, esto es: $Y_1 = 0, \ldots, Y_1 = 40$
La segunda columna corresponde a la alternativa $x_1 = 0$ es una alternativa factible $\forall Y_1$ por lo que se calcula el rendimiento, esto es:

$$c_1x_1 + G_0(Y_0) = (6)(0) + 0 = 0$$

La tercer columna corresponde a la alternativa $x_1 = 1$ es una alternativa factible a partir de $Y_1 = 6$ (Por ser $a_1 = 6$), se calcula el rendimiento, esto es:

$$c_1x_1 + G_0(Y_0) = (6)(1) + 0 = 6$$

Este procedimiento continua hasta la séptima columna que corresponde a la alternativa $x_1 = 6$, ésta es factible a partir de $Y_1 = 36$ (Por ser $a_1 = 6$), se calcula el rendimiento, esto es:

$$c_1x_1 + G_0(Y_0) = (6)(6) + 0 = 36$$

En la penúltima columna se utiliza

$$G_1(Y_1) = max_{x_1}\{c_1x_1 + G_0(Y_0)\}$$

es decir, se obtiene el máximo por fila (por cada valor de Y_1)
En la última columna se anota la alternativa asociada al valor de la columna anterior.
La tabla de la etapa 1 es:

Cuadro 2.10: *Etapa 1 problema de la mochila*

Fuente: Elaboración propia utilizando Excel (2010)

Etapa 1	Alternativas para x_1								
Y_1	0	1	2	3	4	5	6	g*	x_1*
0	0							0	0
1	0							0	0
2	0							0	0
3	0							0	0
4	0							0	0
5	0							0	0
6	0	6						6	1
7	0	6						6	1
8	0	6						6	1
9	0	6						6	1
10	0	6						6	1
11	0	6						6	1
12	0	6	12					12	2
13	0	6	12					12	2
14	0	6	12					12	2
15	0	6	12					12	2
16	0	6	12					12	2
17	0	6	12					12	2
18	0	6	12	18				18	3
19	0	6	12	18				18	3
20	0	6	12	18				18	3
21	0	6	12	18				18	3
22	0	6	12	18				18	3
23	0	6	12	18				18	3
24	0	6	12	18	24			24	4
25	0	6	12	18	24			24	4
26	0	6	12	18	24			24	4
27	0	6	12	18	24			24	4
28	0	6	12	18	24			24	4
29	0	6	12	18	24			24	4
30	0	6	12	18	24	30		30	5
31	0	6	12	18	24	30		30	5
32	0	6	12	18	24	30		30	5
33	0	6	12	18	24	30		30	5
34	0	6	12	18	24	30		30	5
35	0	6	12	18	24	30		30	5
36	0	6	12	18	24	30	36	36	6
37	0	6	12	18	24	30	36	36	6
38	0	6	12	18	24	30	36	36	6
39	0	6	12	18	24	30	36	36	6
40	0	6	12	18	24	30	36	36	6

Etapa 2

Los datos para esta etapa son:

- $c_2 = 16$
- $a_2 = 14$
- $b = 40$

El número de alternativas es:

$$\left[\frac{b}{a_2}\right] = \left[\frac{40}{14}\right] = [2.8571] = 2$$

La ecuación recursiva que se utilizará en esta etapa es:

$$G_2(Y_2) = max_{x_2}\{c_2x_2 + G_1(Y_1)\}$$

$$sujeto\ a$$

$$0 \leq Y_2 \leq 40$$

$$x_2 = 0, 1, \ldots \left[\frac{b}{a_2}\right]$$

También se utilizará la transformación de estados, ésta es:

$$Y_1 = Y_2 - a_2x_2$$

Llenado de la tabla

En la primer columna solo se anotan los valores de Y_2, esto es: $Y_2 = 0, \ldots, Y_2 = 40$
La segunda columna corresponde a la alternativa $x_2 = 0$ es una alternativa factible $\forall Y_2$ pero se utilizará la transformación de estados para poder ligar el beneficio de la etapa anterior y así obtener el rendimiento acumulado, es decir,

$$Y_1 = Y_2 - a_2x_2 \ y \ c_2x_2 + G_1(Y_1)$$

desarrollaremos esto para algunos valores de Y_2.
Como $x_2 = 0 \Rightarrow a_2x_2 = 0 \ \forall Y_2 \ \Rightarrow$ no se utilizarán unidades de la capacidad de la mochila para artículos del tipo 2

- Para $Y_2 = 5$
  Como $a_2x_2 = (14)(0) = 0 \Rightarrow Y_1 = Y_2 - a_2x_2 = 5 - 0 = 5$, entonces se obtiene de la etapa 1 $G_1(Y_1) = G_1(5) = 0$

  $$\Rightarrow c_2x_2 + G_1(Y_1) = (16)(0) + 0 = 0$$

- Para $Y_2 = 10$
  Como $a_2x_2 = (14)(0) = 0 \Rightarrow Y_1 = Y_2 - a_2x_2 = 10 - 0 = 10$, entonces se obtiene de la etapa 1 $G_1(Y_1) = G_1(10) = 6$

  $$\Rightarrow c_2x_2 + G_1(Y_1) = (16)(0) + 6 = 6$$

- Para $Y_2 = 20$
  Como $a_2x_2 = (14)(0) = 0 \Rightarrow Y_1 = Y_2 - a_2x_2 = 20 - 0 = 20$, entonces se obtiene de la etapa 1 $G_1(Y_1) = G_1(20) = 18$

  $$\Rightarrow c_2x_2 + G_1(Y_1) = (16)(0) + 18 = 18$$

- Para $Y_2 = 40$
  Como $a_2x_2 = (14)(0) = 0 \Rightarrow Y_1 = Y_2 - a_2x_2 = 40 - 0 = 40$, entonces se obtiene de la etapa 1 $G_1(Y_1) = G_1(40) = 36$

  $$\Rightarrow c_2x_2 + G_1(Y_1) = (16)(0) + 36 = 36$$

La tercer columna corresponde a la alternativa $x_2 = 1$ es una alternativa factible a partir de $Y_2 = 14$, se desarrollará para algunos valores Y_2 sin olvidar la transformación de estados y la función de rendimiento acumulado.
Como $x_2 = 1 \Rightarrow a_2x_2 = (14)(1) = 14 \Rightarrow$ solo se utilizarán 14 unidades de la capacidad de la mochila para artículos del tipo 2, el resto se complementará con la información de la etapa 1

- Para $Y_2 = 14$
  Como $a_2x_2 = (14)(1) = 14 \Rightarrow Y_1 = Y_2 - a_2x_2 = 14 - 14 = 0$, entonces se obtiene de la etapa 1 $G_1(Y_1) = G_1(0) = 0$

$$\Rightarrow c_2x_2 + G_1(Y_1) = (16)(1) + 0 = 16$$

- Para $Y_2 = 30$
  Como $a_2x_2 = (14)(1) = 14 \Rightarrow Y_1 = Y_2 - a_2x_2 = 30 - 14 = 16$, entonces se obtiene de la etapa 1 $G_1(Y_1) = G_1(16) = 12$

$$\Rightarrow c_2x_2 + G_1(Y_1) = (16)(1) + 12 = 28$$

- Para $Y_2 = 40$
  Como $a_2x_2 = (14)(1) = 14 \Rightarrow Y_1 = Y_2 - a_2x_2 = 40 - 14 = 26$, entonces se obtiene de la etapa 1 $G_1(Y_1) = G_1(24) = 24$

$$\Rightarrow c_2x_2 + G_1(Y_1) = (16)(1) + 24 = 40$$

Este procedimiento continua hasta la cuarta columna que corresponde a la alternativa $x_2 = 2$.
En la penúltima columna se utiliza

$$G_2(Y_2) = max_{x_2}\{c_2x_2 + G_1(Y_1)\}$$

es decir, se obtiene el máximo por fila (por cada valor de Y_2)
En la última columna se anota la alternativa asociada al valor de la columna anterior.
La tabla de la etapa 2 es:

Cuadro 2.11: *Etapa 2 problema de la mochila*

Fuente: Elaboración propia utilizando Excel (2010)

Etapa 2	Alternativas para x_2				
Y_2	0	1	2	g*	x_2*
0	0			0	0
1	0			0	0
2	0			0	0
3	0			0	0
4	0			0	0
5	0			0	0
6	6			6	0
7	6			6	0
8	6			6	0
9	6			6	0
10	6			6	0
11	6			6	0
12	12			12	0
13	12			12	0
14	12	16		16	1
15	12	16		16	1
16	12	16		16	1
17	12	16		16	1
18	18	16		18	0
19	18	16		18	0
20	18	22		22	1
21	18	22		22	1
22	18	22		22	1
23	18	22		22	1
24	24	22		24	0
25	24	22		24	0
26	24	28		28	1
27	24	28		28	1
28	24	28	32	32	2
29	24	28	32	32	2
30	30	28	32	32	2
31	30	28	32	32	2
32	30	34	32	34	1
33	30	34	32	34	1
34	30	34	38	38	2
35	30	34	38	38	2
36	36	34	38	38	2
37	36	34	38	38	2
38	36	40	38	40	1
39	36	40	38	40	1
40	36	40	44	44	2

Etapa 3

Los datos para esta etapa son:

- $c_3 = 10$
- $a_3 = 9$
- $b = 40$

El número de alternativas es:

$$\left[\frac{b}{a_3}\right] = \left[\frac{40}{9}\right] = [4.4444] = 4$$

La ecuación recursiva que se utilizará en esta etapa es:

$$G_3(Y_3) = max_{x_3}\{c_3x_3 + G_2(Y_2)\}$$

$$sujeto\ a$$

$$0 \leq Y_3 \leq 40$$

$$x_3 = 0, 1, \ldots \left[\frac{b}{a_3}\right]$$

La transformación de estados para esta etapa es:

$$Y_2 = Y_3 - a_3x_3$$

Llenado de la tabla

En la primer columna se anotan los valores de Y_3
La segunda columna corresponde a la alternativa $x_3 = 0$ es una alternativa factible $\forall Y_3$ pero se utilizará la transformación de estados para poder ligar el beneficio entre las etapas y obtener el rendimiento acumulado, esto es:

$$Y_2 = Y_3 - a_3x_3\ y\ c_3x_3 + G_2(Y_2)$$

lo desarrollaremos para algunos valores de Y_3.
Si $x_3 = 0$ implica que no se utilizarán unidades de la capacidad de la mochila para artículos del tipo 3

- Para $Y_3 = 20$
  Como $a_3x_3 = (9)(0) = 0 \Rightarrow Y_2 = Y_3 - a_3x_3 = 20 - 0 = 20$, entonces se obtiene de la etapa 2 $G_2(Y_2) = G_2(20) = 22$

  $$\Rightarrow c_3x_3 + G_2(Y_2) = (10)(0) + 22 = 22$$

- Para $Y_3 = 40$
  Como $a_3x_3 = (9)(0) = 0 \Rightarrow Y_2 = Y_3 - a_3x_3 = 40 - 0 = 40$, entonces se obtiene de la etapa 2 $G_2(Y_2) = G_2(40) = 44$

  $$\Rightarrow c_3x_3 + G_2(Y_2) = (10)(0) + 44 = 44$$

La tercer columna corresponde a la alternativa $x_3 = 1$ es una alternativa factible a partir de $Y_3 = 9$, sin olvidar la transformación de estados y la función de rendimiento acumulado.
Lo desarrollaremos para algunos valores de Y_3.
Como $x_3 = 1 \Rightarrow a_3x_3 = (9)(1) = 9\ \Rightarrow$ solo se utilizarán 9 unidades de la capacidad de la mochila para artículos del tipo 3, el resto se complementará con la información de la etapa 1, es decir, con una combinación de artículos del tipo 1 y del tipo 2

- Para $Y_3 = 30$
  Como $a_3x_3 = (9)(1) = 9 \Rightarrow Y_2 = Y_3 - a_3x_3 = 30 - 9 = 21$, entonces se obtiene de la etapa 2 $G_2(Y_2) = G_2(21) = 22$

$$\Rightarrow c_3x_3 + G_2(Y_2) = (10)(1) + 21 = 31$$

- Para $Y_3 = 40$
  Como $a_2x_2 = (9)(1) = 9 \Rightarrow Y_2 = Y_3 - a_3x_3 = 40 - 9 = 31$, entonces se obtiene de la etapa 2 $G_2(Y_2) = G_2(31) = 32$

$$\Rightarrow c_3x_3 + G_2(Y_2) = (10)(1) + 32 = 42$$

Este procedimiento continua hasta la sexta columna que corresponde a la alternativa $x_3 = 4$.
En la penúltima columna se utiliza

$$G_3(Y_3) = max_{x_3}\{c_3x_3 + G_2(Y_2)\}$$

es decir, se obtiene el máximo por fila (por cada valor de Y_3)
En la última columna se anota la alternativa asociada al valor de la columna anterior.
La tabla de la etapa 3 es:

Cuadro 2.12: *Etapa 3 problema de la mochila*

Fuente: Elaboración propia utilizando Excel (2010)

Etapa 3	Alternativas para x_3						
Y_3	0	1	2	3	4	g*	x_3*
0	0					0	0
1	0					0	0
2	0					0	0
3	0					0	0
4	0					0	0
5	0					0	0
6	6					6	0
7	6					6	0
8	6					6	0
9	6	10				10	1
10	6	10				10	1
11	6	10				10	1
12	12	10				12	0
13	12	10				12	0
14	16	10				16	0
15	16	16				16	0
16	16	16				16	0 y 1
17	16	16				16	0 y 1
18	18	16	20			20	2
19	18	16	20			20	2
20	22	16	20			22	0
21	22	22	20			22	0 y 1
22	22	22	20			22	0 y 1
23	22	26	20			26	1
24	24	26	26			26	1 y 2
25	24	26	26			26	1 y 2
26	28	26	26			28	0
27	28	28	26	30		30	3
28	32	28	26	30		32	0
29	32	32	26	30		32	1 y 2
30	32	32	32	30		32	0,1 y 2
31	32	32	32	30		32	0,1 y 2
32	34	32	36	30		36	2
33	34	34	36	36		36	2 y 3
34	38	34	36	36		38	0
35	38	38	36	36		38	0 y 1
36	38	38	38	36	40	40	3
37	38	42	38	36	40	42	1
38	40	42	42	36	40	42	1 y 2
39	40	42	42	42	40	42	1,2 y 3
40	44	42	42	42	40	44	0

Etapa 4

Los datos para esta etapa son:

- $c_4 = 20$
- $a_4 = 17$
- $b = 40$

El número de alternativas es:

$$\left[\frac{b}{a_4}\right] = \left[\frac{40}{17}\right] = [2.3529] = 2$$

La ecuación recursiva que se utilizará en esta etapa es:

$$G_4(Y_4) = max_{x_4}\{c_4x_4 + G_3(Y_3)\}$$

$$sujeto\ a$$

$$0 \leq Y_4 \leq 40$$

$$x_4 = 0, 1, \ldots \left[\frac{b}{a_4}\right]$$

También se utilizará la transformación de estados para esta etapa.

$$Y_3 = Y_4 - a_4x_4$$

Llenado de la tabla

En la primer columna se anotan los valores de Y_4
A continuación llenaremos la columna 4, esta corresponde a la alternativa $x_4 = 2 \Rightarrow$ $a_4x_4 = (17)(2) = 34 \Rightarrow$ se utilizarán 34 unidades de la capacidad de la mochila para artículos del tipo 4, el resto se complementará con la información de la etapa 3, es decir, con una combinación de artículos del tipo $1, 2\ y\ 3$
Esta alternativa es factible para $Y_4 = 34$ en adelante, entonces, desarrollaremos:

- Para $Y_4 = 37$
  Como $a_4x_4 = (17)(2) = 34 \Rightarrow Y_3 = Y_4 - a_4x_4 = 37 - 34 = 3$, entonces se obtiene de la etapa 3 $G_3(Y_3) = G_3(3) = 0$

$$\Rightarrow c_4x_4 + G_3(Y_3) = (20)(2) + 0 = 40$$

- Para $Y_4 = 40$
  Como $a_4x_4 = (17)(2) = 34 \Rightarrow Y_3 = Y_4 - a_4x_4 = 40 - 34 = 6$, entonces se obtiene de la etapa 3 $G_3(Y_3) = G_3(6) = 6$

$$\Rightarrow c_4x_4 + G_3(Y_3) = (20)(2) + 6 = 46$$

En la penúltima columna se utiliza

$$G_4(Y_4) = max_{x_4}\{c_4x_4 + G_3(Y_3)\}$$

es decir, se obtiene el máximo por fila (por cada valor de Y_4)
En la última columna se anota la alternativa asociada al valor de la columna anterior.
La tabla de la etapa 4 es:

Cuadro 2.13: *Etapa 4 problema de la mochila*

Fuente: Elaboración propia utilizando Excel (2010)

Etapa 4	Alternativas para x_4				
Y_4	0	1	2	g*	x_4*
0	0			0	0
1	0			0	0
2	0			0	0
3	0			0	0
4	0			0	0
5	0			0	0
6	0			0	0
7	6			6	0
8	6			6	0
9	6			6	0
10	10			10	0
11	10			10	0
12	10			10	0
13	12			12	0
14	12			12	0
15	16			16	0
16	16			16	0
17	16	20		20	1
18	16	20		20	1
19	20	20		20	0 y 1
20	20	20		20	0 y 1
21	22	20		22	0
22	22	20		22	0
23	22	26		26	1
24	26	26		26	0 y 1
25	26	26		26	0 y 1
26	26	30		30	1
27	28	30		30	1
28	30	30		30	0 y 1
29	32	32		32	0 y 1
30	32	32		32	0 y 1
31	32	36		36	1
32	32	36		36	1
33	36	36		36	0 y 1
34	36	36	40	40	2
35	38	40	40	40	1 y 2
36	38	40	40	40	1 y 2
37	40	42	40	42	1
38	42	42	40	42	0 y 1
39	42	42	40	42	0 y 1
40	42	46	46	46	1 y 2

Para recuperar la solución utilizaremos el siguiente algoritmo

1. $n =$ *número de artículos*
2. Hacer $q = n$
3. b es la capacidad de la mochila disponible, nos fijamos en la mejor alternativa de la tabla en la etapa q, en la última columna, en la fila b; guardamos x_q

4. Realizamos la transformación de estados

$$Y_{q-1} = Y_q - a_q x_q$$

hacer $b = Y_{q-1}$

5. Hacer $q = q - 1$.

   - Si $q \geq 1$ ir a 3,
   - Si $q = 0$ terminar la solución corresponde al conjunto de decisiones $x_{q's}$ de cada iteración y el beneficio óptimo es el colocado en la última tabla, penúltima columna, fila b

Apliquemos el algoritmo para recuperar la solución: $n = 4 \Rightarrow q = 4$
$b = 40$
La decisión de la etapa 4 en la fila 40 no es única, entonces, la solución no es única

- $x_4 = 1 \Rightarrow Y_3 = 40 - 17 = 23 \Rightarrow b = 23$
- $x_4 = 2 \Rightarrow Y_3 = 40 - 34 = 6 \Rightarrow b = 6$

Hacemos $q = 3$ seguimos con los dos casos:

- Si $b = 23$, nos fijamos en la tabla de la etapa 3 en la fila 23 y la decisión óptima es $x_3 = 1 \Rightarrow Y_2 = 23 - 9 = 14 \Rightarrow b = 14$
- Si $b = 6$, nos fijamos en la tabla de la etapa 3 en la fila 6 y la decisión óptima es $x_3 = 0 \Rightarrow Y_2 = 6 - 0 = 6 \Rightarrow b = 6$

Ahora $q = 2$ los dos casos son:

- Si $b = 14$, nos fijamos en la tabla de la etapa 2 en la fila 14 y la decisión óptima es $x_2 = 1 \Rightarrow Y_1 = 14 - 14 = 0 \Rightarrow b = 0$
- Si $b = 6$, nos fijamos en la tabla de la etapa 2 en la fila 6 y la decisión óptima es $x_2 = 0 \Rightarrow Y_1 = 6 - 0 = 6 \Rightarrow b = 6$

$q = 1$

- Si $b = 0$, nos fijamos en la tabla de la etapa 1 en la fila 0 y la decisión óptima es $x_1 = 0 \Rightarrow Y_1 = 0 - 0 = 0 \Rightarrow b = 0$
- Si $b = 6$, nos fijamos en la tabla de la etapa 1 en la fila 6 y la decisión óptima es $x_1 = 1 \Rightarrow Y_1 = 6 - 6 = 0 \Rightarrow b = 0$

Como $b = 0$ terminamos, las soluciones óptimas son: $X_1^* = (0, 1, 1, 1)$ y $X_2^* = (1, 0, 0, 2)$ con $z(X^*) = 46$

La programación dinámica evita enlistar todas las combinaciones de las alternativas y además elimina las soluciones no factibles, ya que en cada etapa se optimiza el subproblema sobre sus alternativas las cuales son decisiones factibles. Por otro lado todas las soluciones no óptimas en cada etapa también son descartadas, por

lo que se concluye que con la programación dinámica se reduce el número de combinaciones de alternativas para poder llegar a la solución óptima del problema de optimización.

La programación dinámica proporciona todas las soluciones del problema en el caso de que la solución no sea única, además si la última tabla se desarrolla para todos los valores de Y_n es posible hacer un análisis de sensibilidad, esto es:
Si por alguna razón la capacidad de la mochila es menor a la b inicial, por ejemplo si necesitamos llevar un artículo no previsto con un peso de 8 unidades, recuperamos la solución con el mismo algoritmo, pero iniciando con $b = 32$

- $q = 4$ $b = 32$ la decisión óptima es $x_4 = 1$
- $q = 3$ $b = 32 - 17 = 15$ la decisión óptima es $x_3 = 0$
- $q = 2$ $b = 15 - 0 = 15$ la decisión óptima es $x_2 = 1$
- $q = 1$ $b = 15 - 14 = 1$ la decisión óptima es $x_1 = 0$

La solución óptima es: $X^* = (0, 1, 0, 1)$ y $z(X^*) = 36$

Capítulo 3

NP-Completez

Cuando necesitamos resolver un problema de optimización combinatoria lo que se desea es obtener una solución factible tal que optimice la función objetivo en un tiempo razonable, en algunos casos es posible obtenerlo de manera rápida por medio de algún algoritmo exacto en un tiempo polinomial, pero en algunos casos el tiempo para obtener dicha solución crece de manera exponencial y esto hace que el problema se vuelva intratable.

Durante muchos años se han desarrollado diversos métodos para resolver problemas combinatorios, pero hasta ahora no se ha encontrado algún algoritmo que resuelva todas las instancias de un problema en particular en tiempo polinomial, por esta razón, se han desarrollado los algoritmos heurísticos y metaheurísticos que proporcionan una solución factible en un tiempo "razonable" pero en muchos casos no óptima.

La teoría de la NP-completez empezó a desarrollarese en 1970, esta teoría clasifica a los problemas de decisión como difíciles o fáciles de resolver.

Stephen Cook en 1970 publicó el artículo *The Complexity of Theorem-Proving Procedures*, en él demostró que el problema de Satisfacibilidad (SAT) es NP-completo, éste fue el primer problema que se demostró pertenecía a este conjunto.
En 1972 Richard Karp publicó el artículo *Reductibility among combinatorial problem*, donde catalogó a 20 problemas como NP-completos a partir del problema de SAT.
Antes de empezar con la teoría de la NP-completez, se proporcionarán algunas definiciones importantes.

3.1. Definiciones [Papadimitriou, C.(1998)]

Definición 7 *Un problema de decisión es una pregunta que sólo puede ser contestada con una respuesta afirmativa (sí) o negativa (no).*

Observación 10 *A todo problema de optimización se le asocia un problema de decisión.*

Para ejemplificar la observación anterior se enlistan algunos problemas de decisión:

- Problema del clique: Dada una gráfica $G = [X, A]$ y n un número entero, ¿Existe un clique de tamaño n? (Un clique es una gráfica completa)
- Problema del circuito hamiltoniano: Sea $G = [X, A]$ una gráfica, ¿Existe un circuito en G que visite todos los nodos exactamente una vez?
- Problema del acoplamiento máximo: Sea $G = [X, A]$ una gráfica conexa y n un número entero, ¿Existe un acoplamiento en G con n o más aristas?
- Problema de cobertura de nodos: Dada $G = [X, A]$ y n un número entero, ¿Existe un conjunto C de n vértices tal que para todo arco de G es adyacente al menos a un nodo de C?
- Problema del conjunto independiente: Dada $G = [X, A]$ y n un número entero, ¿Existe un conjunto I de n vértices tal que 2 nodos en I nos están conectados por alguna arista en A(G)?
- Problema 3-acoplamiento: Dados 3 conjuntos U, V, W tal que $|U| = |V| = |W|$ y M un subconjunto de $U \times V \times W$, ¿Existe $M' \subseteq M$ con $|M'| = |U|$ tal que (u, v, w) y (u', v', w') son tercias diferentes en M' (es decir, $u \neq u', v \neq v', w \neq w'$)
- Problema 3-recubrimiento: Dado una familia $F = \{S_1, S_2, ..., S_n\}$ de n subconjuntos de $S = \{u_1, u_2, ..., u_3\}$ cada uno de cardinalidad 3 ¿Existe una subfamilia de m subconjuntos que cubra a S?
- Problema de la mochila binario: $(a_i = c_i)$ Dados los enteros c_j para $j = 1, 2, ..., n$ y k ¿Existe un subconjunto S de $\{1, 2, ..., n\}$ tal que $\Sigma_{i \in S} c_j = k$
- Problema de la mochila general: Dados los enteros c_j para $j = 1, 2, ..., n$ y k ¿Existen enteros $x_j \geq 0$ para $j = 1, 2, ..., n$ tal que $\Sigma_{i=1}^{n} c_i x_i \leq k$?

Para poder catalogar a los problemas de decisión como fáciles o difíciles, se darán algunas definiciones necesarias.

Definición 8 *Un algoritmo es un conjunto ordenado y finito de operaciones que permiten solucionar un problema [Vélez y Montoya (2007)].*

Un algoritmo se puede medir con respecto al número de operaciones aritméticas que deben realizarse para solucionar un problema.

Un problema es fácil si existe un algoritmo que lo resuelve en tiempo polinomial; es decir, si el número de operaciones necesarias para que el algoritmo resuelva el problema es una función polinomial del tamaño del problema. Si esta función no es polinomial, se dice que el algoritmo es no polinomial y el problema se considera difícil. [Vélez y Montoya (2007)]

Definición 9 *La clase P contiene a los problemas de decisión que pueden ser resueltos por un algoritmo, el cual siempre finaliza con una respuesta (sí o no) en cierto número de pasos, con una duración polinomial, es decir, en tiempo polinomial.*

Definición 10 *En la clase NP no se mide cuanto tiempo tarda el algoritmo en dar una respuesta, ya sea afirmativa o negativa, si no el tiempo que tardamos en verificar la validez de la respuesta, las iniciales NP denotan nondeterministic polinomial.*

Definición 11 *Reducción polinomial. Sean Q_1 y Q_2 dos problemas de decisión, se dice que Q_1 se reduce en tiempo polinomial a Q_2 si y sólo si se cumple alguna de las siguientes condiciones:*

- *Se encuentra un algoritmo A_1 (De tiempo polinomial) que transforme el problema Q_1 a el problema Q_2*
- *Existe un algoritmo A_1 de tiempo polinomial para Q_1 que utilice varios pasos o subrutinas de un algoritmo A_2, donde A_2 es un algoritmo que es utilizado para resolver el problema Q_2*

Si Q_1 se reduce polinomialmente a Q_2 (en tiempo polinomial) se denota como $Q1 \ \alpha \ Q2$

Definición 12 *Un problema Q es NP-Completo si:*

- *Q es NP.*
- *Además todos los problemas que son NP se reducen en tiempo polinomial a Q.*

Observación 11 *Cuando un problema es catalogado como NP-Completo no existen métodos eficaces para obtener la solución óptima, es decir, para llegar a ésta nos tomaría mucho tiempo (tiempo exponencial) y en muchos casos no se llegaría al valor óptimo sino a una aproximación, por esta razón existen los métodos heurísticos y metaheurísticos los cuales son procedimientos simples que a menudo son dirigidos por el sentido común o por un patrón, en muchos casos la solución encontrada por alguno de estos métodos no llega a ser la óptima si no una aproximación, se podría decir que los métodos heurísticos y metaheurísticos sacrifican optimalidad por rapidez.*

Observación 12 *La propiedad de reducción polinomial es transitiva y reflexiva, esto es:*

- *Si P_1 se reduce polinomialmente a P_2 y P_2 se reduce polinomialmente a P_3, entonces P_1 se reduce polinomialmente a P_3*
- *Si P_1 se reduce polinomialmente a P_2, entonces P_2 se reduce polinomialmente a P_1.*

Con estas propiedades podemos llegar a las características de los problemas NP-completos:

1. Un problema NP-Completo no puede ser resuelto por un algoritmo polinomial conocido.
2. Si existe un algoritmo polinomial para algún problema que sea NP-Completo, entonces éste es para todos los problemas que sean NP-Completos.

Observación 13 *Si un problema de decisión es catalogado como NP-Completo, entonces el problema de optimización asociado a él es NP-duro.*

3.2. Reducciones Polinomiales

Las reducciones polinomiales para llegar al problema de la mochila serán:

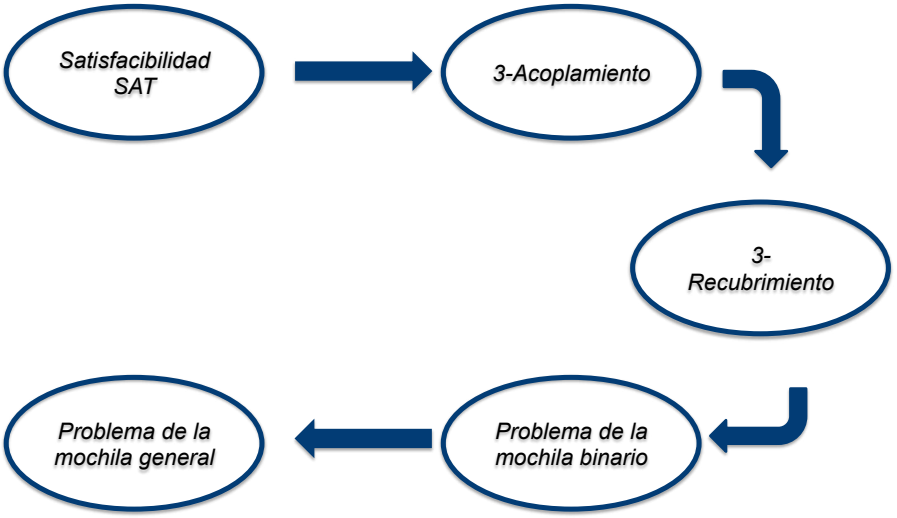


Figura 3.1: *Reducciones Polinomiales para llegar al problema de la mochila*

Fuente: Elaboración propia utilizando Geogebra 5 (2016)

3.2.1. Problema del SAT α Problema 3-Acoplamiento

Demostración

El problema de 3-acoplamiento es NP ya que con un algoritmo no determinístico sólo se necesitan de $q = |U|$ pasos para verificar en tiempo polinomial que ninguna de estas tercias coinciden, por lo que para demostrar que es NP-completo hace falta encontrar una reducción polinomial de un problema que es NP-completo a éste, se partirá del problema de Satisfactibilidad (SAT).

Problema del SAT: Define un conjunto de variables booleanas sean $\{x_1, x_2, ..., x_n\}$, el complemento de cada una de ellas se denota $\overline{x_i}$; las variables $\bar{x}_{i's}, x_{i's}$ forman cláusulas $C_1, C_2, ..., C_r$ tal que C_i es de la siguiente forma $C_i = (x_1 \vee x_3 \vee \bar{x}_2)$, con estas cláusulas se forma E tal que $E = C_1 \wedge C_2 \wedge ... \wedge C_r$, para que exista una asignación de las variables $\bar{x}_{i's}, x_{i's}$ con 0 y 1 tal que E sea verdadera, cada C_i debe ser verdadera, la pregunta es: ¿Existe una asignación de las variables para que E sea veradadera?

Algunos ejemplos de enunciados del SAT son:

Ejemplo 10 $E = (\bar{x}_1 \vee \ \overline{x}_2) \wedge (\ x_1) \wedge (\ \bar{x}_2)$. Con la siguiente asignación $\bar{x}_2 = 1$ y $x_1 = 1$, $\bar{x}_1 = 0, x_2 = 0$, el enunciado es verdadero.

Ejemplo 11 $E = (\bar{x}_1 \vee \bar{x}_2) \wedge (x_1) \wedge (x_2)$. Este enunciado es falso ya que si $x_2 = 1$ y $x_1 = 1$, entonces $\bar{x}_1 = 0$ y $\bar{x}_2 = 0$ por lo que no se cumpliría la cláusula 1.

Problema 3-acoplamiento: Dados 3 conjuntos U, V, W con la misma cardinalidad y M un subconjunto de $U \times V \times W$, ¿Existe M' un subconjunto de M con $|M'| = |U|$ tal que (u, v, w) y (u', v', w') son tercias diferentes en M' (es decir, $u \neq u', v \neq v'$ y $w \neq w'$).

Transformación del Problema SAT al 3-acoplamiento

Sea E una fórmula Booleana que contiene las cláusulas $C_1, C_2, ..., C_m$ las cuales se forman a partir de una o algunas de las variables originales o el complemento de éstas, es decir, $x_1, x_2, ..., x_n$; o $\bar{x_1}, \bar{x_2}, ..., \bar{x_n}$, por lo que E es de la siguiente forma $E = C_1 \wedge C_2 \wedge, ..., \wedge C_m$ y cada cláusula es de la forma $C_i = x_r \vee \overline{x_s}$.

De E se puede construir una gráfica $G = [X, A]$ donde $X = U \times V \times W$ (con $|U| = |V| = |W|$); los conjuntos de vértices o nodos son generados por las variables y cláusulas de E, las adyacencias se dan de acuerdo a las cláusulas. Utilizando G se podrá concluir lo siguiente: El enunciado E es verdadero, si y sólo si, existe en G un 3- acoplamiento perfecto M.
El problema de 3-acoplamiento se puede definir con el enfoque de gráficas esto es:
Sea $G = [X, A]$ una gráfica, M un 3-acoplamiento perfecto en G si $|M| = |U|$ y sus elementos son ternas (u, v, w) tal que $u \in U$, $v \in V$ y $w \in W$ y además $(u, v, w) \neq (u', v', w')$ con $u \neq u', v \neq v'$ y $w \neq w'$, es decir, que existe un conjunto de triángulos que no son adyacentes entre sí y que contienen todos los nodos de la gráfica.
Para construir G, se partirá de un enunciado del SAT ($E = C_1 \wedge C_2 \wedge, ..., \wedge C_m$) donde E contiene m cláusulas y n variables, entonces, los conjuntos de nodos se definen de la siguiente manera:

- U : Consta de todas las variables de cada una de las cláusulas de la fórmula booleana, aunque no estén explícitamente en ellas, es decir, si en E está x_i en la cláusula j pero no está $\bar{x}_i$; en U si estará contenida. $U = \{x_i^j, \bar{x}_i^j | i = 1, 2, ..., n; j = 1, 2; ..., m\}$ los elementos de E se denotan:

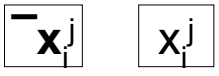


Figura 3.2: *Nodos contenidos en el conjunto U*

Fuente: Elaboración propia utilizando PowerPoint (2010)

- V : Contiene tres tipos de nodos
  $V_1 = \{a_i^j | i = 1, 2, ..., n; j = 1, 2, ..., m\}$
  $V_2 = \{v_j | j = 1, 2, ..., m\}$
  $V_3 = \{c_i^j | i = 1, 2, ..., n-1; j = 1, 2, ..., m\}$
  $V = V_1 \bigcup V_2 \bigcup V_3$ y se denota

Figura 3.3: *Nodos contenidos en el conjunto V*

Fuente: Elaboración propia utilizando PowerPoint (2010)

- W : También contiene tres clases de nodos
  $W_1 = \{b_i^j | i = 1, 2, ..., n; j = 1, 2, ..., m\}$
  $W_2 = \{w_j | j = 1, 2, ..., m\}$
  $W_3 = \{d_i^j | i = 1, 2, ..., n-1; j = 1, 2, ..., m\}$
  $W = W_1 \bigcup W_2 \bigcup W_3$ y se denota

Figura 3.4: *Nodos contenidos en el conjunto W*

Fuente: Elaboración propia utilizando PowerPoint (2010)

Proposición 2 *Los conjuntos U, V y W tienen la misma cardinalidad (*$|U| = |V| = |W|$*)*
Demostración: *Como E contiene m cláusulas y n variables; si contamos las* $x_i^j, \bar{x}_i^j$*, entonces* $|U| = 2nm$*:*
Por otro lado $V = V_1 \bigcup V_2 \bigcup V_3$ *por lo que* $|V| = |V_1| + |V_2| + |V_3|$*:*
Por como están definidos los conjuntos V_1, V_2 *y* V_3 *las cardinalidades de cada uno de ellos son:*
$|V_1| = nm$
$|V_2| = m$
$|V_3| = m(n-1)$
Entonces $|V| = |V_1| + |V_2| + |V3| = nm + m + m(n-1) = m(n+1+n-1) = 2nm$

Análogamente para el conjunto W:
Como $W = W_1 \bigcup W_2 \bigcup W_3$ *por lo que*
$|W| = |W_1| + |W_2| + |W_3| = nm + m + m(n-1) = m(n+1+n-1) = 2nm$
$\therefore |U| = |V| = |W|$

Las adyacencias constarán de tres tipos de ternas, éstas son:

- Adyacencias del tipo 1 A_1:Todas las variables originales y sus complementos $(x_i^j, \bar{x}_i^j)$ formarán las tercias con los nodos de la forma a_i^j, b_i^j
  $A_1 = \{(a_i^j, b_i^j, x_i^j)$ para $i = 1, 2, ...n; j = 1, 2, ...m\} \bigcup \{(a_i^{j+1}, b_i^j, \bar{x}_i^j)$ para $i = 1, 2, ..., n; j = 1, 2, ...m$ donde $a_i^{m+1} = a_i^1\}$

- Adyacencias del tipo 2 A_2: Los nodos v_j y w_j formarán las tercias con las variables originales o con su complemento $(x_i^j,\ \bar{x}_i^j)$; siempre y cuando dicha variable aparezca en la cláusula j, es decir,
  $A_2 = \{(v_j, w_j, x_i^j)$ *si y sólo si* $x_i \in C_j\} \bigcup \{(v_j, w_j, \bar{x}_i^j)$ *si y sólo si* $\bar{x}_i \in C_j\}$.

- Adyacencias del tipo 3 A_3: Estas se darán con las variables que sobran de A_2, es decir, si la variable q-esíma aparece en la cláusula r-esíma, esto quiere decir que existirá en A_2 la tercia $(v_r, w_r x_q^r)$; si además la variable q-esíma aparece también en la cláusula t-esíma, entonces la tercia $(v_t, w_t, x_q^{r)} \in A_2$, se hace el mismo razonamiento si fuera $\bar{x}_q^j$ la que aparece en la cláusula r y t: Utilizando esto, las tercias del tipo tres serán:
  Como x_q $(\bar{x}_q)$ aparece en las cláusulas t y r, esto quiere decir que x_q^r y x_q^t $(\bar{x}_q^r, \bar{x}_q^t)$ son adyacentes con sus respectivas v_j y w_j, por lo que las demás variables de la forma $x_q^j(\bar{x}_q^j)$ para $j \neq t$ y $j \neq r$ formarán las tercias con c_k^j y d_k^j para $j \neq t$ y $j \neq r$; es decir,
  $A_3 = \{(c_k^j, d_k^j, x_q^j)$ *si y sólo si* $j \neq t$ *y* $j \neq r\} \bigcup \{(c_k^j, d_k^j, \bar{x}_q^j)$ *si y sólo si* $j \neq t$ *y* $j \neq r\}$ para $i = 1, ..., n;\ j = 1, ..., m$ y $k = 1, ..., m(n-1)$:

Algunos ejemplos para construir la gráfica G:

Ejemplo 12 *Sea* $E = (\bar{x}_1 \vee \bar{x}_2) \wedge (x_1) \wedge (\bar{x}_2)$
Entonces los conjuntos de nodos son:

- $U = \{x_1^1, \bar{x}_1^1, x_1^2, \bar{x}_1^2, x_1^3, \bar{x}_1^3, x_2^1, \bar{x}_2^1, x_2^2, \bar{x}_2^2, x_2^3, \bar{x}_2^3\}$

- $V = \{a_1^1, a_1^2, a_1^3, a_2^1, a_2^2, a_2^3, v_1, v_2, v_3, c_1^1, c_1^2, c_1^3\}$

- $W = \{b_1^1, b_1^2, b_1^3, b_2^1, b_2^2, b_2^3, w_1, w_2, w_3, d_1^1, d_1^2, d_1^3\}$

Las adyacencias se ilustran en las siguiente gráfica:

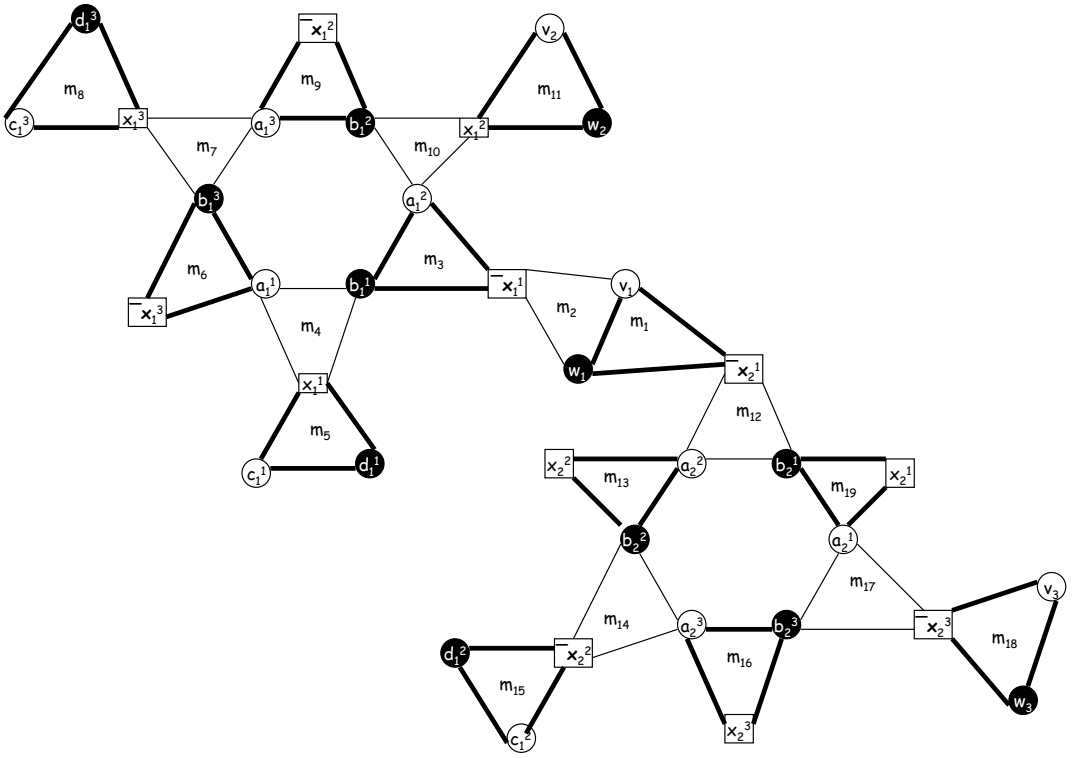


Figura 3.5: *3 - Acoplamiento Perfecto*

Fuente: Elaboración propia utilizando Geogebra 5 (2016)

Ejemplo 13 *Sea* $E = (\bar{x}_1 \vee \bar{x}_2) \wedge (x_1) \wedge (x_2)$*, la gráfica correspondiente a este enunciado es:*

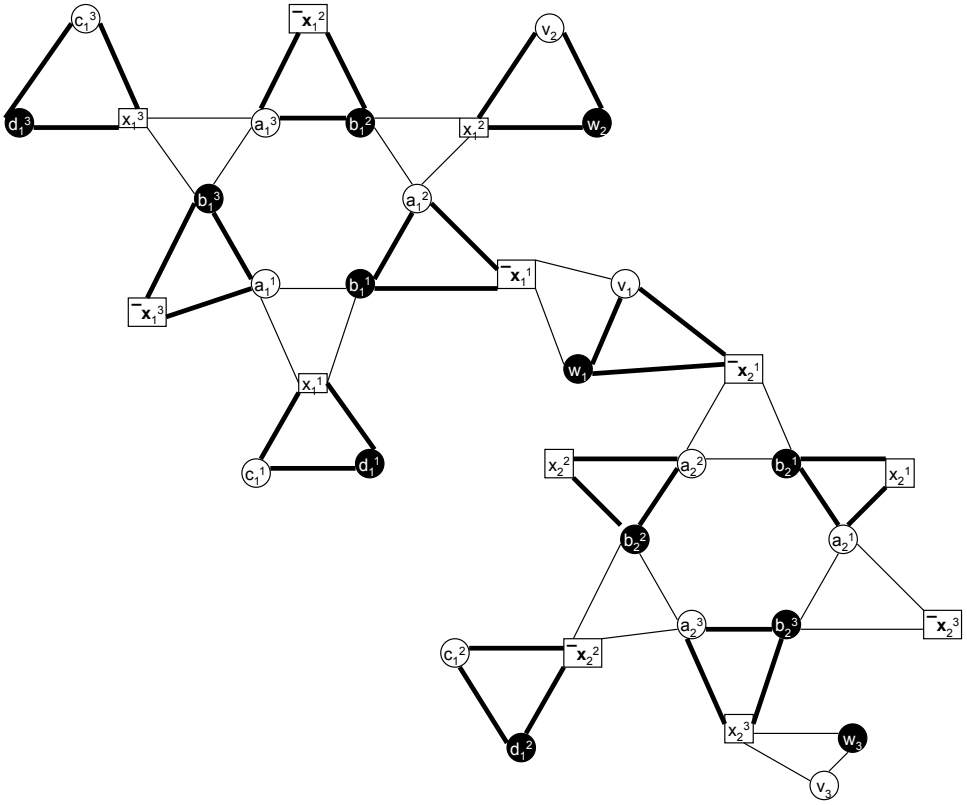


Figura 3.6: *3 - Acoplamiento*

Fuente: Elaboración propia utilizando Geogebra 5 (2016)

$\Longrightarrow$ *Sean* $G = [X, M]$ *donde* $X = U \bigcup V \bigcup W$ *y* $M = A_1 \bigcup A_2 \bigcup A_3$ *y* M' *un* 3-acoplamiento perfecto, es decir, $M' \subseteq M$ *y* $|M'| = |U|$
P.D. El enunciado del problema del SAT correspondiente a la gráfica G es verdadero.
Cada estrella equivale a una variable en el enunciado.
El número de vértices v_j o w_j ($|V_2| = |W_2|$) es el número de cláusulas en E.
Todas las tercias del tipo 2 en M definen al enunciado del SAT. Utilizando el ejemplo 1: La gráfica G contiene 2 estrellas, por lo que el enunciado del SAT tendrá 2 variables.
$|V_2| = 3 = |W_2|$; esto quiere decir, que E contiene 3 cláusulas.
Para formar las cláusulas se analizarán las tercias del tipo 2, estas son:
$\{(v_1, w_1, \bar{x}_1^1), (v_1, w_1, \bar{x}_2^1), (v_2, w_2, x_1^2), (v_3, w_3, \bar{x}_2^3)\}$
como el subíndice es el número de variable y el hiper-índice es la cláusula, el enunciado correspondiente a esta gráfica es: $E = (\bar{x}_1 \vee \bar{x}_2) \wedge (x_1) \wedge (\bar{x}_2)$; para dar la asignación de las variables se utilizará el 3-acoplamiento perfecto.
Como M' es un 3-acoplamiento perfecto, entonces, M' contiene a todos los nodos de G, es decir, contiene a todos los $v_{j's}$ y $w_{j's}$. Como las únicas tercias que se forman con estos nodos son las de tipo 2, entonces, $(v_j, w_j, x_i^j) \in M$ o $(v_j, w_j, \bar{x}_i^j) \in M'$, esto

quiere decir, que en la estrella correspondiente a la variable i las tercias que están en M' son del tipo 1, en otras palabras, si $(v_j, w_j, x_i^j) \in M'$, entonces, en la estrella i $(a_i^{j+1}, b_i^j, \bar{x}_i^j) \in M'$ y si $(v_j, w_j, \bar{x}_j^i) \in M'$, entonces, $(a_j, b_j, x_j) \in M'$, esto hace que la variable x_i sólo tome un valor (0 o 1).
Por ser un 3-acoplamiento perfecto implica que no va haber intersección entre las tercias del tipo 1 y tipo 2 en M', por lo que las tercias del tipo 2 dan la asignación para las variables que aparecen en cada una de las cláusulas del enunciado.
En el ejemplo 1, las tercias del tipo 2 que están en M' son:
$\{(v_2, w_2, x_1^2), (v_1, w_1, \bar{x}_2^1), (v_3, w_3, \bar{x}_2^3)\}$, esto implica que:

- Como x_1^2 está en la primera tercia; entonces $x_1 = 1$ en la cláusula 2.
- Como $\bar{x}_2^1$ está en la segunda tercia, entonces $x_2 = 0$ en la cláusula 2 y $\bar{x}_2 = 1$
- Como $\bar{x}_2^3$ está en la tercera tercia; entonces $x_2 = 0$ en la cláusula 3 y $\bar{x}_2 = 1$

Por lo que la asignación para las dos variables que aparecen en el enunciado es: $x_1 = 1$ y $x_2 = 0$.
Por lo tanto, si G tiene un 3-acoplamiento entonces el enunciado relacionado a la gráfica G es verdadero.

Observación 14 *Si G no tiene un 3-acoplamiento perfecto, entonces existen tercias del tipo 2 que no pertenecen M', por lo que no se tiene una asignación para la variable en esa cláusula, por lo tanto, E sería falso.*

$\Longleftarrow$ Se parte de un enunciado $E = C_1 \wedge C_2 \wedge, ... \wedge C_m (con\ C_j = x_i \vee \bar{x}_k)$
P. D. Si E es verdadero, entonces existe un 3-acoplamiento perfecto M' en $G = [X, M]$ donde $X = U \bigcup V \bigcup W$ y $M = A_1 \bigcup A_2 \bigcup A_3$.
Se contruirá el acoplamiento de la siguiente manera:
Se empieza con $M' = \emptyset$
Las primeras tercias que se incluirán en M' son las tercias del tipo 2; en una estrella puede haber más de una tercia de este tipo, si este es el caso, entonces éstas deben contener a la variable original o a su complemento, si contienen a ambas, entonces debe estar ligada con otra estrella donde las tercias del tipo 2 sólo estén formadas por las variables originales o su complemento. Se pueden dar, entonces los siguientes casos:

- **Caso1)**: Sea $x_i = 1$ en la asignación; esto quiere decir, que las primeras tercias que formarán parte de M' son (v_j, w_j, x_i^j) tal que hacen que el enunciado sea verdadero), las segundas tercias que formarán parte de M'; son las del tipo 1 $(a_i^{j+1}, b_i^j, \bar{x}_i^j)$ y las últimas tercias son todas las del tipo 3.
- **Caso 2)** Sea $x_i = 0$ en la asignación; esto quiere decir, que las primeras tercias que formarán parte de M' son $(v_j, w_j, \bar{x}_i^j)$ (tal que hacen que el enunciado sea verdadero), las segundas tercias que formarán parte de M'; son las del tipo 1 (a_i^j, b_i^j, x_i^j) y las últimas tercias son todas las del tipo 3.

P.D. $|M'| = |U| = 2mn$.
El número de tercias del tipo 2 que forman parte de M' son m ya que se tienen m cláusulas verdaderas.
El número de tercias del tipo 1 que forman parte de M' son $\frac{|U|}{2}$ ya que o son las tercias formadas por las variables originales o sus complementos, por lo tanto son $\frac{2nm}{2} = nm$.
El número de tercias del tipo 3 que están en M' es $|c_k^j| = |d_k^j| = m(n-1)$.
Por lo tanto $|M'| = m + nm + m(n-1) = m(1+n+n-1) = 2mn$.
Por lo tanto es un 3-acoplamiento perfecto.

3.2.2. Problema 3-Acoplamiento α Problema 3-Recubrimiento

Recordemos los problemas:
Problema 3-recubrimiento
Dado una familia $F = \{S_1, S_2, ..., S_n\}$ de n subconjuntos de $S = \{u_1, u_2, ..., u_{3r}\}$ cada uno de cardinalidad 3 ¿Existe una subfamilia de m subconjuntos que cubra a S (cubra a todos los nodos de G)?
Problema 3-acoplamiento
Dados 3 conjuntos U, V, W con la misma cardinalidad y M un subconjunto de $U \times V \times W$ ¿Existe M' un subconjunto de M con $|M| = |U|$ tal que (u, v, w) y (u', v', w') son tercias diferentes en M' (es decir, $u \neq u', v \neq v', w \neq w'$) **Demostración**
El problema de 3-recubrimiento es NP ya que con un algoritmo no determinístico se necesitan de r pasos para verificar que F' ($|F'| = r$) cubre a todo S.

Transformación del problema 3-acoplamiento al problema 3-recubrimiento

Para reducir el problema de 3-acoplamiento al 3-recubrimiento se definirán los conjuntos de cada problema de la siguiente manera:
Sean:

- $U \bigcup V \bigcup W = S$ como $|U| = |V| = |W| = 2nm$, entonces $|S| = |U| + |V| + |W| = 2nm + 2nm + 2nm = 3(2nm) = 3r$.
- $T = U \times V \times W = F$ donde los elementos de F son tercias de la forma (u, v, w) por lo tanto $|S_i| = 3$ para $i = 1, 2, ..., n$.

$\Longrightarrow$ P. D. Si el problema de 3-recubrimiento es verdadero, entonces, el problema de 3-acoplamiento es verdadero.
Como el problema de 3- recubrimiento es verdadero, entonces existe un subconjunto de F, sea F' tal que $|F| = r$ y este cubre a todo S.
Como en F' son tercias y $|F'| = r = |U| = |V| = |W|$ además F' cubre a todo S, es decir, cubre a $U \bigcup V \bigcup W$: Podemos definir $F' = M'$ y M' es un 3-acoplamiento.
$\Longleftarrow$ P. D. Si el problema de 3-acoplamiento es verdadero, entonces, el problema de

3-recubrimiento es verdadero.
Como el problema de $3-$ *acoplamiento* es verdadero, entonces, para 3 conjuntos U, V *y* W tal que $|U| = |V| = |W|$, sea $T = U \times V \times W$ existe un subconjunto $M' \subseteq T$ tal que $|M'| = |U|$ y los elementos de M son tercias diferentes $(u, v, w) \neq (u', v', w')$ si y sólo si $u \neq u',\ v \neq v'$ *y* $w \neq w'$
Si $M' = F'$, entonces, F' contiene tercias tal que $|F'| = r$ y F' cubre a todo S, ya que M' es un 3-acoplamiento.
Por lo tanto el problema 3-recubrimiento es NP- completo.
Para ejemplificar esto tomamos el ejemplo anterior donde E es verdadera.
$S = U \bigcup V \bigcup W$ donde $|S| = 36$
F son todas las tercias en G $|F| = 19$,
La subfamilia de F que cubre a todo S es M' donde $|M'| = 12$. Con esto se cumple que $|S| = 36 = 3r = 3(12)$; se tienen r elementos de F tal que cubran a S. Como M' es un 3-acoplamiento M' cubre a todos los elementos de S.

3.2.3. Problema 3-Recubrimiento α Problema de la mochila binario

Para esta demostración se utilizará una versión especial del problema de la mochila binario, donde $c_j = a_j$.
Dados los enteros c_j para $j = 1, 2, ..., n$ y k ¿Existe un subconjunto S de $\{1, 2, ..., n\}$ tal que $\Sigma_{i \in S} c_j = k$
Demostración:
El problema binario de la mochila es NP ya que con un algoritmo no determinístico se puede verificar en n pasos que $\sum_{j \in S} c_j = k$

Transformación del Problema 3-recubrimiento al problema de la mochila binario

Para reducir el problema de 3-recubrimiento al de la mochila se hará la siguiente transformación:
Empezamos con una familia F de n conjuntos de cardinalidad 3, de estos se pueden definir los enteros $c_1, c_2, ..., c_n$y k.
Se puede suponer que todos los conjuntos en F son vectores binarios de longitud $3r$, por ejemplo, si $3r = 12$, entonces las tercias $\{u_1, u_5, u_6\}$ y $\{u_2, u_4, u_6\}$ se convierten en 000000110001 y 000000101010, respectivamente. Si interpretamos estos vectores binarios como enteros escritos en base $(n+1)$, los enteros correspondientes al conjunto S_j son: $c_j = \sum_{u_t \in S_j} (n+1)^{l-1}$
donde l son los lugares donde están los unos.
Además, sea K el entero correspondiente a $1111, ..., 11$ ($3r$ unos)
$K = \sum_{j=0}^{3r-1} (n+1)^j$
Para ejemplificar esto, sea G la siguiente gráfica:

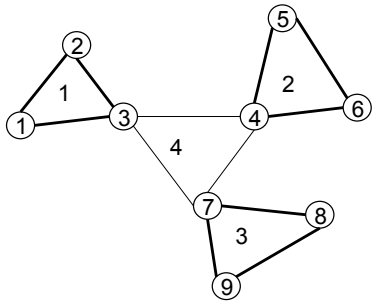


Figura 3.7: *3-Recubrimiento*

Fuente: Elaboración propia utilizando Geogebra 5 (2016)

donde $n = 4$ $3r = 9$ y $r = 3$
Los números enteros $c_{j's}$ se construyen a partir de todas la tercias en G, es decir, se tendrán 4 $c_{j's}$ éstos se obtienen de la siguiente manera:

- De la tercia 1 (u_1, u_2, u_3) se construye el siguiente bit-vector 000000111, entonces,
  $c_1 = 5^0 + 5^1 + 5^2 = 31$
- De la tercia 2 (u_4, u_5, u_6) se construye el siguiente bit-vector 000111000, entonces,
  $c_2 = 5^3 + 5^4 + 5^5 = 3875$
- De la tercia 3 (u_7, u_8, u_9) se construye el siguiente bit-vector 111000000, entonces,
  $c_3 = 5^6 + 5^7 + 5^8 = 484375$
- De la tercia 4 (u_3, u_4, u_7)se construye el siguiente bit-vector 001001100, entonces,
  $c_4 = 5^2 + 5^3 + 5^6 = 15775$
- Donde $k = 111111111$, entonces,
  $k = \sum_{j=0}^{3r-1}(n+1)^j = 5^0 + 5^1 + 5^2 + 5^3 + 5^4 + 5^5 + 5^6 + 5^7 + 5^8 = 488281$.

Con estas conversiones, el problema de la mochila asociado a la gráfica anterior es:

$$Maxz = 31x_1 + 3875x_2 + 484375x_3 + 15775x_4$$
$$sujeto\ a$$
$$31x_1 + 3875x_2 + 484375x_3 + 15775x_4 \leq 488281$$

$\Longrightarrow$ Supongamos que la respuesta al problema de la mochila es verdadera.
P. D. El problema de 3-recubrimiento es verdadero.
Como la respuesta al problema de la mochila es verdadera implica que, dados los enteros c_j; $j = 1, 2, ..., n$ y k, entonces existe un subconjunto S' de $\{1, 2, ..., n\}$ tal que

$$\sum_{j \in S'} c_j = k.$$

Realizando la suma aritmética en base $(n+1)$, es decir, $k = \sum_{j=0}^{3r-1}(n+1)^j$ se puede notar que existe exactamente un 1 en cada una de las $3r$ posiciones y esto equivale a que $C = \{S_j : j \in S'\}$ cubre exactamente a $S = \{u_1, u_2, ..., u_{3r}\}$.

Siguiendo el ejemplo, las variables que hacen verdadero el problema de la mochila son: x_1, x_2 yx_3, cuyos coeficientes son c_1, c_2 yc_3 y sus respectivos bit-vectores son :

- $c_1 \rightarrow 000000111$
- $c_2 \rightarrow 000111000$
- $c_3 \rightarrow 111000000$

Como cada entrada de los bit-vectores representa un nodo de la gráfica, entonces queda cubierto el conjunto de los 9 vértices.
$\Longleftarrow$ Supongamos que el problema de 3-recubrimiento es verdadero.
P. D. El problema de la mochila es verdadero.
Como el problema de 3- recubrimiento sea verdadero, esto implica que, dado una familia $F = \{S_1, S_2, ..., S_n\}$ de n subconjuntos de $S = \{u_1, u_2, ..., u_{3r}\}$ cada uno de cardinalidad 3, existe una subfamilia de r subconjuntos (F') que cubra a S.
Haciendo la transformación anterior, el subconjunto F' que cubre a todo S son las tercias que generarán a los $c_{j's} \in S'$ ya que, como F' cubre a toda S
$\Rightarrow \sum_{j \in S'} c_j = \sum_{u_j \in S_j}(n+1)^{l-1} = \sum_{j=0}^{3r-1}(n+1)^j = k$
Por lo tanto la respuesta al problema de la mochila es verdadero.
$\therefore$ El problema de la mochila binario es NP-completo.

3.2.4. Problema de la mochila binarioα Problema de la mochila general

Demostración:
El problema de decisión de la mochila general dice: Dados los enteros c_j para $j = 1, 2, ..., n$ y k ¿Existen enteros $y_j \geq 0$ para $j = 1, 2, ..., n$ tal que $\Sigma_{i=1}^{n} a_i y_i \leq k$?
El problema de la mochila general es NP ya que es un caso especial del problema de programación lineal entera. (PPLE Dada una matriz de enteros A de $m \times n$ y un vector de m enteros b, ¿Existe un n-vector de enteros Y tal que $AY = b$, para $Y \geq 0$?)

Transformación del Problema de la mochila binario al Problema de la mochila general

Para demostrar que es NP-completo, primero se transformará el problema de la mochila binario al problema general de la mochila.
Sean $\{c_1, c_2, ..., c_n, k\}$ los coeficientes enteros del problema binario de la mochila, podemos suponer que $1 \leq c_j \leq k$ para $j = 1, 2, ..., n$ y $k > 0$
Se definirán los coeficientes del problema de la mochila general $\{d_1, d_2, ..., d_{2n}, L\}$ de la siguiente manera:

Sean:

- $d_j = \begin{cases} M^{n+1} + M^j + c_j & j \leq n \\ M^{n+1} + M^{j-n} & en\ otro\ caso \end{cases}$
- $L = nM^{n+1} + \sum_{j=1}^{n} M^j + k$
- $M = 2n(n+1)k$

Para ilustrar esta transformación se utilizará el siguiente problema de la mochila binario:

$$\begin{gathered} Maxz = 2x_1 + x_2 \\ sujeto\ a \\ 2x_1 + x_2 \leq 3 \\ x_i \in \{0, 1\} \end{gathered}$$

$n = 2$ y $k = 3$
Sustituyendo para obtener M, d_j y k
$M = 2$ x 2 x 3 x $3 = 36$
para $j = 1, 2.$
$d_1 = 36^3 + 36^1 + 2 = 46694$
$d_2 = 36^3 + 36^2 + 1 = 47953$
para $j = 3, 4.$
$d_3 = 36^3 + 36^1 = 46692$
$d_4 = 36^3 + 36^2 = 47952$
y
$L = 2$ x $36^3 + 36^1 + 36^2 + 3 = 94647$
El problema general de la mochila tendría la siguiente forma:

$$\begin{gathered} Maxz = 46694y_1 + 47953y_2 + 46692y_3 + 47952y_4 \\ sujeto\ a \\ 46694y_1 + 47953y_2 + 46692y_3 + 47952y_4 \leq 94647 \\ y_i \in Z\ y_i \geq 0 \end{gathered}$$

$\Longrightarrow$ Supongamos que existe una solución para el problema de la mochila general. Es decir, sean $\{y_1.y_2, ..., y_{2n}\}$ tal que $\sum_{j=1}^{2n} d_j y_j = L$; en otras palabras sustituyendo: $\sum_{j=1}^{n}(M^{n+1} + M^j + c_j)y_j + \sum_{j=n+1}^{2n}(M^{n+1} + M^{j-n})y_j = nM^{n+1} + \sum_{j=1}^{n} M^j + k$
entonces,
$M^{n+1} \sum_{j=n+1}^{2n} y_j + \sum_{j=1}^{n} M^j(y_j + y_{j+n}) + \sum_{j=1}^{n} c_j y_j = nM^{n+1} + \sum_{j=1}^{n} M^j + k.$
Sean $\sum_{j=1}^{2n} y_j = n$ y $y_j + y_{j+n} = 1$, entonces,
$M^{n+1}n + \sum_{j=1}^{n} M^j + \sum_{j=1}^{n} c_j y_j - nM^{n+1} - \sum_{j=1}^{n} M^j = k$
Por lo que, $\sum_{j=1}^{n} c_j y_j = k.$
Por lo tanto $Y = (y_1, y_2, ..., y_{2n})$ es una solución para el problema binario.
$\therefore$ La respuesta al problema de la mochila binario es verdadera.

$\Longleftarrow$ Supongamos que existe una solución para el problema de la mochila binario. Es decir, sean $\{x_1, x_2, ..., x_n\}$ tal que $\sum_{j=1}^{n} c_j x_j = k.$

P.D.Existen $\{y_1, y_2, ..., y_{2n}\}$ tal que $\sum_{j=1}^{2n} d_j y_j = L$
Sean $y_j = x_j$ para $j = 1, ..., n$ y $y_j = 1 - x_{j-n}$ para $j = n+1, ..., 2n$, entonces,
$\sum_{j=1}^{2n} d_j y_j = \sum_{j=1}^{n} d_j x_j + \sum_{j=n+1}^{2n} d_j(1 - x_{j-n}) =$
$\sum_{j=1}^{n}(M^{n+1} + M^j + c_j)x_j + \sum_{j=n+1}^{2n}(M^{n+1} + M^{j-n})(1 - x_{j-n}) =$
$\sum_{j=1}^{2n} M^{n+1}(x_j + 1 - x_{j-n}) + \sum_{j=1}^{n} M^j x_j + \sum_{j=1}^{n} c_j x_j + \sum_{j=n+1}^{2n} M^{j-n}(1 - x_{j-n}) =$
$nM^{n+1} + \sum_{j=1}^{n} M^j + k = L$
Por lo tanto la respuesta al problema de la mochila general es verdadero.
$\therefore$ El problema de la mochila es NP-completo.

Capítulo 4

Métodos heurísticos y metaheurísticos

Los algoritmos heurísticos son una herramienta útil para resolver problemas de optimización. Los problemas de optimización se dividen en dos categorías: Problemas con variables continuas y problemas con variables discretas. Ambas clases tienen problemas que se dificultan, ya sea por la complejidad de la función a optimizar o por que el tiempo para encontrar el óptimo crece exponencialmente y hace que los problemas se vuelvan intratables, por esta razón los algoritmos heurísticos han tenido un desarrollo importante, ya que éstos son rápidos y en algunos casos son eficientes al llegar a una solución, que si no es la óptima, es una buena aproximación. En toda esta búsqueda, para tratar de mejorar las aproximaciones que se obtienen con los métodos heurísticos tradicionales, surgen las técnicas metaheurísticas, que tratan de dales inteligencia a estos algoritmos.

4.1. Heurísticos

El término heurístico en los problemas de optimización surge al tratar de resolver problemas reales usando el conocimiento disponible (usando la experiencia) de manera inteligente.
La palabra *heurístico* proviene de la palabra *"heurískein"* que significa encontrar o descubrir. [De Antonio (2011)]

Definición 13 *Un heurístico es un procedimiento simple que a menudo está basado en el sentido común, que proporciona una buena solución (aunque no necesariamente la óptima) a problemas difíciles, de un modo fácil y rápido, los problemas no necesariamente deben ser combinatorios.*

Otra definición de métodos heurísticos (aplicados a la Investigación de Operaciones) según [Melián, B. (2003)] es: *Se califica de heurístico a un procedimiento para el que se tiene un alto grado de confianza en que encuentra soluciones de alta calidad con un costo computacional razonable, aunque no se garantice su optimalidad o su*

factibilidad, e incluso, en algunos casos, no se llegue a establecer lo cerca que se está de dicha situación. Se usa el calificativo heurístico en contraposición a exacto
Los métodos heurísticos principalmente se utilizan porque:

- Hasta este momento no se conoce algún método exacto para resolver el problema.
- Se conoce algún método exacto, pero el tiempo computacional es muy costoso.
- Permiten incorporar nuevas condiciones (restricciones) en el modelo matemático.
- Proporciona una buena solución inicial de manera rápida.

4.1.1. Clasificación de los heurísticos

[Martí, R. (2003)] Los métodos heurísticos resuelven solamente los problemas para los que fueron creados, aunque es posible generalizar algunos pasos, y es necesario conocer a la perfección el problema a resolver para adaptar el heurístico.
Por esta razón es difícil dar una clasificación de los métodos heurísticos que incluyan a todos, ya que en ocasiones surgen a partir de un problema específico, una de las clasificaciones que incluye a los heurísticos más conocidos es la siguiente:

a) **Métodos Constructivos:** Construyen paso a paso la solución del problema. Usualmente son deterministas ya que en cada iteración eligen a la mejor opción. Un ejemplo de esta heurística son los algoritmos voraces o ”greedy” El más conocido es el algoritmos de Kruskal el cual obtiene el árbol expandido de peso mínimo en una gráfica conexa.

b) **Métodos de descomposición** Sigue la filosofía "divide y vencerás". Descompone el problema original en problemas más sencillos de resolver, tomando en cuenta que los subproblemas deben pertenecer al mismo problema.

c) **Métodos de manipulación del modelo (relajación):** Obtienen una solución del problema original a partir de la obtenida de un problema relajado. (con menos restricciones)

d) **Metodos de reducción:** Identifica condiciones que se cumplen en el proceso de obtención de buenas soluciones que permiten simplificar el tratamiento del problema utilizándolas como restricciones del problema.

e) **Métodos de búsqueda por entornos (por vecindades):** Comienza con una solución inicial, ésta se modifica sucesivamente para obtener una solución final que es la mejor encontrada hasta ese momento.

Un algoritmo heurístico se puede catalogar como bueno si cumple con las siguientes propiedades:

- Eficiente. Debe terminar con una solución con un esfuerzo computacional realista.
- Bueno. La solución debe ser "muy cercana al óptimo".
- Robusto. La probabilidad de obtener una mala solución (lejos del óptimo) debe ser baja.

En toda esta búsqueda de mejorar las soluciones obtenidas en los heurísticos aparecen los algoritmos metaheurísticos.

4.2. Metaheurísticos

El término metaheurístico apareció por primera vez en el artículo sobre diferentes caminos para resolver problemas de programación entera, donde incluye la búsqueda tabú de [Glover, F. (1986)].
Las técnicas metaheurísticas son procedimientos de búsqueda que parten de un punto inicial o un conjunto de puntos iniciales (una población) que usualmente no es óptima, a partir de ella(s) se generan soluciones vecinas, de entre las cuales se elige una(s) que satisfacen algún criterio, a partir de la cual comienza de nuevo el proceso. El proceso se detiene cuando se satisface un criterio establecido previamente.

Definición 14 *Una metaheurística se refiere a un método que guía y modifica otras heurísticas para producir soluciones más allá de las que normalmente se generan en un heurístico de búsqueda (que generalmente es búsqueda local). La heurística guiada por tal meta-estrategia pueden ser procedimientos de alto nivel que crean un proceso capaz de escapar de los óptimos locales y realizar una búsqueda robusta de un espacio de solución. [Melián, B. (2003)]*

Según [Melián, B.(2003)] existen ciertas propiedades deseables que se busca tenga un metaherústico, algunas de estas son:

- Simple y coherente
- Efectiva, eficaz y eficiente: Esto quiere decir que la solución obtenida debe ser de alta calidad, que la probabilidad de obtener soluciones óptimas debe ser alta y que el algoritmo debe tener un buen aprovechamiento computacional (en tiempo y memoria)
- Debe ser general, adaptable e interactivo, es decir, puede ser aplicado prácticamente a cualquier problema de optimización, además deben estar abiertos a que el usuario pueda hacer cualquier modificación como: incorporar alguna restricción al problema o cambiar algún criterio según el conocimiento del usuario.

Las principales características de las metaheurísticas [Rodríguez, K.,(2018)] son:

- Son ciegas, no identifican si han llegado al óptimo, por esta razón es necesario tener un criterio para detener las iteraciones.

- Son algoritmos aproximativos: No garantizan la optimalidad y la eficiencia de la solución en muchos casos no es posible determinarla ya que no se conoce el óptimo para la instancia, aunque es posible que con la definición de las vecindades y una buena solución inicial, se tenga una mejor aproximación.

- Aceptan en ocasiones malos movimientos, por lo general con base a la función objetivo; esto ayuda a no caer en óptimos locales. Incluso es posible aceptar una solución no factible para acceder a regiones no exploradas con anterioridad.

- Son relativamente sencillos.

- Son generales, prácticamente se pueden aplicar a cualquier problema que se desee optimizar, la técnica será eficiente si los criterios seleccionados son adecuados al problema particular.

- La regla de selección depende del momento del proceso y una memoria hasta ese momento.

4.2.1. Clasificación de los metaheurísticos

Las metaheurísticas se pueden clasificar de a cuerdo a la metodología en la que se basan, esto es:

a) **De relajación:** Utilizan procedimientos de relajación del modelo original, esto hace que el problema sea más fácil de resolver, cuya solución facilita la solución del problema original.

b) **Constructivos**: Se orientan a los procedimientos que tratan de la obtención de una solución a partir del análisis y selección paulatina de las componentes que la forman.

c) **De búsqueda**: Guían su búsqueda a travez de transformaciones o movimientos para recorrer el espacio de soluciones y así explotar las estructuras de sus entornos.
   Los algoritmos de búsqueda tienen dos características, estas son:

   - Intensificación: Esto es explorar las buenas y "malas"soluciones encontradas.
   - Diversificación: Explora diferentes regiones para mejorar la solución encontrada hasta este momento y en ocaciones explora regiones que empeoran la mejor solución.

   Si se logran equilibrar estos dos puntos, ayuda a que el algoritmo identifique rápidamente soluciones de buena calidad, además de que consume menos tiempo para llegar a la mejor solución.

d) **Evolutivas**: Están enfocadas a los procedimientos basados en conjuntos de soluciones que evolucionan sobre el espacio de soluciones.

4.3. Calidad de los algoritmos heurísticos y metaheurísticos

La calidad de un algoritmo se puede medir al comparar la solución óptima de la instancia que se está resolviendo con la solución obtenida con el heurístico, pero existen casos en los que la solución óptima de la instancia no se conoce, entonces, se puede comparar con una cota que debe ser de buena calidad, esto es: [Martí, R. (2003)]

- **Si se conoce la solución óptima de la instancia que se está resolviendo:**
  Sean
  Z_o el valor de la función objetivo de la solución óptima,.
  Z_h el valor de la función objetivo de la solución obtenida con el heurístico aplicado a la misma instancia.
  La desviación porcentual de la solución heurística con respecto a la óptima es:

$$Desviación\ de\ la\ solución = (\frac{Z_o - Z_h}{Z_o})100$$

  El porcentaje de la efectividad de la solución heurística con respecto a la óptima es:

$$\%\ de\ efectividad = 100 - (\frac{Z_o - Z_h}{Z_o})100$$

- **Si no se conoce la solución óptima de la instancia que se está resolviendo:** Podemos encontrar una cota para el valor de la función objetivo para así compararlo con la solución proporcionada por el heurístico, esto puede ser mediante:
  - Un algoritmo exacto truncado: Consiste en aplicar el algoritmo exacto como ramificación y acotamiento y después de determinado número de iteraciones se considera la mejor solución encontrada hasta ese momento para compararla son la solución que proporciona el heurístico o metaheurístico.
  - Comparar la solución obtenida con otro método heurístico o metaheurístico.

4.4. Algoritmo heurístico constructivo

Los algoritmos constructivos se desarrollan para cada problema particular, ya que utiliza las características de cada uno para construir la solución, en cada iteración toma una decisión de tal manera que se escoge siempre la mejor. En la siguiente sección se desarrollará un algoritmo glotón para el problema de la mochila

4.4.1. Algoritmo voraz para el problema de la mochila

La característica que nos ayudará a construir la solución de un problema tipo mochila es que *solo tiene una restricción* y gracias a esto podemos obtener un beneficio marginal unitario mediante los cocientes $\frac{c_i}{a_i}$, esto nos ayudará a identificar de qué artículos se obtiene mayor beneficio, por lo general la función objetivo de un problema tipo mochila es a maximizar por lo que los cocientes se ordenarán de manera decreciente y se van metiendo artículos de acuerdo a la capacidad disponible, en el siguiente diagrama de flujo se explican con detalle los pasos:

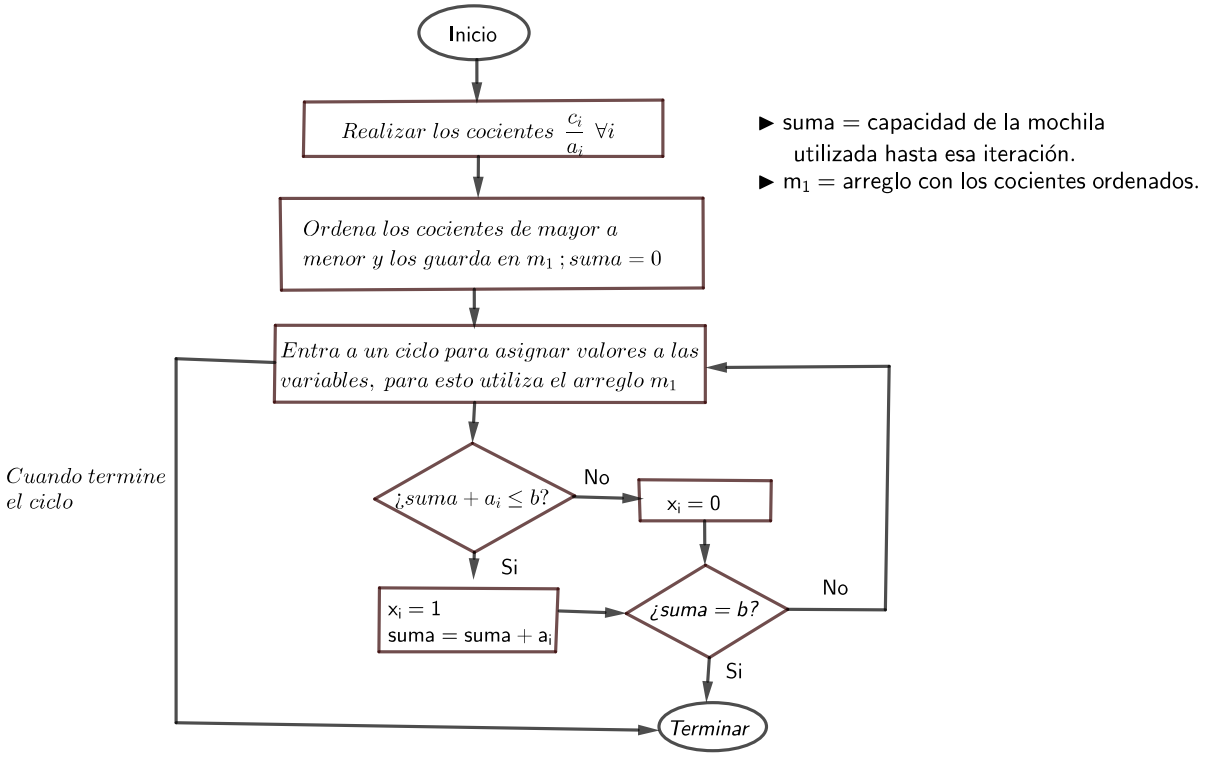


Figura 4.1: *Heurístico constructivo para el problema binario*

Fuente: Elaboración propia utilizando Geogebra 5 (2016)

En las siguientes secciones se explicará con detalle el algoritmo glotón para diferentes problemas de la mochila.

Problema de la mochila general

Propósito: Obtener una solución factible para el problema de la mochila general con capacidad b; el beneficio y el peso para el artículo i son: c_i y a_i respectivamente.

Paso 1. **Inicialización**
$X = (0, ..., 0)$; $z(X) = 0$; $b = b$; $j = 0$;

Paso 2. Obtener los cocientes $\frac{c_i}{a_i}$ y ordenarlos de mayor a menor; sean $x_{1'}, x_{2'}, ..., x_{n'}$ las variables con el nuevo orden, sea $j = 1'$ ir a 3

Paso 3. Si $\frac{b}{a'_j} \in Z$, entonces $x_j = \frac{b}{a'_j}$ ir a 4
Si $\frac{b}{a'_j} \notin Z$, entonces $x_j = [\frac{b}{a'_j}]$ ir a 4

Paso 4. Hacer $b = b - [x_j * a'_j]$; $j = (j + 1)'$
Si $b > 0$ y $j \leq n$; ir al paso 3
Si $b = 0$ o $j > n$ terminar; la solución encontrada es óptima.

Ejemplo 14 *Considera el siguiente problema de la mochila*

$$Max z = 6x_1 + 16x_2 + 10x_3 + 20x_4$$
$$sujeto\ a$$
$$6x_1 + 14x_2 + 9x_3 + 17x_4 \leq 40$$
$$x_i \geq 0\ y\ x_i \in Z\ para\ i = 1, ..., 4$$

Vamos a resolver este problema utilizando el algoritmo constructivo y compararemos la solución obtenida con la solución óptima: $X = (1, 0, 0, 2)$ *y* $z(X) = 46$
Obteniendo los cocientes y ordenando:

Cuadro 4.1: *Cocientes ordenados, problema de la mochila general(1)*

Orden inicial i	1	2	3	4
$\frac{c_i}{a_i}$	$\frac{6}{6} = 1$	$\frac{16}{14} = 1.14$	$\frac{10}{9} = 1.11$	$\frac{20}{17} = 1.17$
Orden final	4	2	3	1

Iteración 1
$b = 40$, $j = 0$, $X = (0, 0, 0, 0)$, $Z(X) = 0$
El artículo del que se obtiene mayor beneficio es el 4, *entonces* $x_4 = [\frac{40}{17}] = [2.35] = 2$
$b = 40 - 34 = 6$; $j = 2$
Iteración 2 *El segundo artículo con mayor beneficio es el* 2, *entonces* $x_2 = [\frac{6}{14}] = [0.42] = 0$
$b = 6 - 0 = 6$; $j = 3$
Iteración 3 *El tercer artículo con mayor beneficio es el* 3, *entonces* $x_3 = [\frac{6}{9}] = [0.66] = 0$
$b = 6 - 0 = 6$; $j = 4$
Iteración 4 *El cuarto artículo con mayor beneficio es el* 1, *entonces* $x_1 = [\frac{6}{6}] = [1] = 1$
$b = 6 - 6 = 0$; $j = 5$
El criterio de parada se cumple, entonces la solución es:

$$X = (1, 0, 0, 2)\ con\ Z(X) = 46$$

que coincide con la solución óptima, pero no siempre se corre con la misma suerte.

Ejemplo 15 *Considera el siguiente problema de la mochila*

$$Maxz = 14x_1 + 25x_2 + 50x_3$$
$$sujeto\ a$$
$$3x_1 + 5x_2 + 7x_3 \leq 13$$
$$x_i \geq 0\ y\ x_i \in Z\ para\ i = 1, ..., 3$$

Lo resolveremos utilizando el algoritmo constructivo y compararemos la solución obtenida con la solución óptima: $X = (2, 0, 1)$ *y* $z(X) = 78$

Cuadro 4.2: *Cocientes ordenados, problema de la mochila general (2)*

Orden inicial i	1	2	3
$\frac{c_i}{a_i}$	$\frac{14}{3} = 4.66$	$\frac{25}{5} = 5$	$\frac{50}{7} = 7.14$
Orden final	3	2	1

Iteración 1
$b = 13$, $j = 0$, $X = (0, 0, 0)$, $Z(X) = 0$
El artículo del que se obtiene mayor beneficio es el 3, *entonces* $x_3 = [\frac{13}{7}] = [1.85] = 1$
$b = 13 - 7 = 6$; $j = 2$
Iteración 2 *El segundo artículo con mayor beneficio es el* 2, *entonces* $x_2 = [\frac{6}{5}] = [1.2] = 1$
$b = 6 - 5 = 1$; $j = 3$
Iteración 3 *El tercer artículo con mayor beneficio es el* 1, *entonces* $x_1 = [\frac{1}{3}] = [0.33] = 0$
$b = 6 - 0 = 6$; $j = 4$
El criterio de parada se cumple, entonces la solución es:

$$X = (0, 1, 1)\ con\ Z(X) = 75$$

si comparamos con la solución óptima: $X = (2, 0, 1)$ *y* $z = 78$, *concluimos que es una aproximación a la solución óptima.*

Utilizando estos resultados podemos obtener la efectividad del algoritmo.

Problema de la mochila binario

Este algoritmo es análogo al anterior, la diferencia es que cada variable tendrá una cota, esta es: $x_i \leq 1\ \forall\ i$
Propósito: Obtener una solución factible para el problema de la mochila binario con capacidad b; los costos y pesos para el artículo i son: c_i y a_i respectivamente.

Paso 1. **Inicialización**
$X = (0, ..., 0)$, $z(X) = 0$, $j = 0$, $b = b$

Paso 2. Obtener los cocientes $\frac{c_i}{a_i}$ y ordenar los cocientes de mayor a menor; sean $x'_1, x'_2, ..., x'_n$ las variables con el nuevo orden, sea $j = 1'$ ir a 3

Paso 3. Si $\frac{b}{a'_j} \geq 1$, entonces $x_j = 1$ ir a 4
Si $\frac{b}{a'_j} < 1$, entonces $x_j = 0$ ir a 4

Paso 4. Hacer $b = b - [x_j * a'_j]$; $j = (j+1)'$
Si $b > 0$ y $j \leq n$; ir al paso 3
Si $b = 0$ o $j > n$ terminar; la solución encontrada es óptima.

Para probar este algoritmo y obtener su efectividad, se utilizarán las siguientes instancias: (obtenidas en: *https://people.sc.fsu.edu/ jburkardt/datasets/knapsack*$_0$1/*knapsack*$_0$1.*html*) 5 *artículos.*

$$max\ z = 24x_1 + 13x_2 + 23x_3 + 15x_4 + 16x_5$$
$$sujeto\ a$$
$$12x_1 + 7x_2 + 11x_3 + 8x_4 + 9x_5 \leq 26$$
$$x_i \in \{0,1\}\ \forall i$$
$$\text{Óptimo } X = (0,1,1,1,0)\ z(X) = 51$$

Cuadro 4.4: *Cocientes ordenados, problema de la mochila binario (1)*

Orden inicial	1	2	3	4	5
$\frac{c_i}{a_i}$	$\frac{24}{12} = 2$	$\frac{13}{7} = 1.8571$	$\frac{23}{11} = 2.0909$	$\frac{15}{8} = 1.875$	$\frac{16}{9} = 1.7777$
Orden final	2	4	1	3	5

Cuadro 4.5: *Iteraciones del algoritmo constructivo aplicado al problema de la mochila binario (1)*

# Iteración	b	Es ¿$\frac{b}{a_{i'}} < 1$? o ¿$\frac{b}{a_{i'}} \geq 1$?	Valor de x'_i
1	$b = 26$	$\frac{26}{11} > 1$	$x_3 = 1$
2	$b = 26 - 11 = 15$	$\frac{15}{12} > 1$	$x_1 = 1$
3	$b = 15 - 12 = 3$	$\frac{3}{8} < 1$	$x_4 = 0$
4	$b = 3 - 0 = 3$	$\frac{3}{7} < 1$	$x_2 = 0$
5	$b = 3 - 0 = 3$	$\frac{3}{9} < 1$	$x_5 = 0$

La solución heurística es: $X^* = (1,0,1,0,0)$ con $z(X^*) = 47$

6 artículos.

$$maz\ z = 50x_1 + 50x_2 + 64x_3 + 46x_4 + 50x_5 + 5x_6$$
$$sujeto\ a$$
$$56x_1 + 59x_2 + 80x_3 + 64x_4 + 75x_5 + 17x_6 \leq 190$$
$$x_i \in \{0,1\}\ \forall i$$
$$\text{Óptimo } X = (1,1,0,0,1,0)\ z(X) = 150$$

Al obtener los cocientes:

Cuadro 4.6: *Cocientes ordenados, problema de la mochila binario (2)*

Orden inicial	1	2	3	4	5	6
$\frac{c_i}{a_i}$	$\frac{50}{56}$ 0.8928	$\frac{50}{59}$ 0.8474	$\frac{64}{80}$ 0.75	$\frac{46}{64}$ 0.7187	$\frac{50}{75}$ 0.6666	$\frac{5}{17}$ 0.29941
Orden final	1	2	3	4	5	6

Cuadro 4.7: *Iteraciones del algoritmo constructivo aplicado al problema de la mochila binario (2)*

# Iteración	b	Es ¿$\frac{b}{a_{i'}} < 1$? o ¿$\frac{b}{a_{i'}} \geq 1$?	Valor de x'_i
1	$b = 190$	$\frac{190}{56} > 1$	$x_1 = 1$
2	$b = 190 - 56 = 134$	$\frac{134}{59} > 1$	$x_2 = 1$
3	$b = 134 - 59 = 75$	$\frac{75}{80} < 1$	$x_3 = 0$
4	$b = 75 - 0 = 75$	$\frac{75}{64} > 1$	$x_4 = 1$
5	$b = 75.64 = 11$	$\frac{11}{75} < 1$	$x_5 = 0$
6	$b = 11 - 0 = 11$	$\frac{11}{17} < 1$	$x_6 = 0$

La solución heurística es: $X^* = (1,1,0,1,0,0)$ con $z(X^*) = 146$

7 artículos.

$$maz\ z = 70x_1 + 20x_2 + 39x_3 + 37x_4 + 7x_5 + 5x_6 + 10x_7$$
$$\text{sujeto a}$$
$$31x_1 + 10x_2 + 20x_3 + 19x_4 + 4x_5 + 3x_6 + 6x_7 \leq 50$$
$$x_i \in \{0,1\}\ \forall i$$
$$\text{Óptimo } X = (1,0,0,1,0,0,0)\ z(X) = 107$$

Al obtener los cocientes:

Cuadro 4.8: *Cocientes ordenados, problema de la mochila binario (3)*

Orden inicial	1	2	3	4	5	6	7
$\frac{c_i}{a_i}$	$\frac{70}{31}$ 2.258	$\frac{20}{10}$ 2	$\frac{39}{20}$ 1.95	$\frac{37}{19}$ 1.9473	$\frac{7}{4}$ 1.75	$\frac{5}{3}$ 1.666	$\frac{10}{6}$ 1.666
Orden final	1	2	3	4	5	6	7

:

Cuadro 4.9: *Iteraciones del algoritmo constructivo aplicado al problema de la mochila binario (3)*

# Iteración	b	Es ¿$\frac{b}{a_{i'}} < 1$? o ¿$\frac{b}{a_{i'}} \geq 1$?	Valor de x_i'
1	$b = 50$	$\frac{50}{31} > 1$	$x_1 = 1$
2	$b = 50 - 31 = 19$	$\frac{19}{10} > 1$	$x_2 = 1$
3	$b = 19 - 10 = 9$	$\frac{9}{20} < 1$	$x_3 = 0$
4	$b = 9 - 0 = 9$	$\frac{9}{19} < 1$	$x_4 = 0$
5	$b = 9 - 0 = 9$	$\frac{9}{4} > 1$	$x_5 = 1$
6	$b = 9 - 4 = 5$	$\frac{5}{3} > 1$	$x_6 = 1$
7	$b = 5 - 3 = 2$	$\frac{2}{6} < 1$	$x_7 = 0$

La solución heurística es: $X^* = (1, 1, 0, 0, 1, 1, 0)$ con $z(X^*) = 102$

8 artículos.

$$maz\ z = 350x_1 + 400x_2 + 450x_3 + 20x_4 + 70x_5 + 8x_6 + 5x_7 + 5x_8$$
$$sujeto\,a$$
$$25x_1 + 35x_2 + 45x_3 + 5x_4 + 25x_5 + 3x_6 + 2x_7 + 2x_8 \leq 104$$
$$x_i \in \{0, 1\}\ \forall i$$
$$\text{Óptimo } X = (1, 0, 1, 1, 1, 0, 1, 1)\ z(X) = 900$$

Obtenemos los cocientes

Cuadro 4.10: *Cocientes ordenados, problema de la mochila binario (4)*

Orden inicial	1	2	3	4	5	6	7	8
$\frac{c_i}{a_i}$	$\frac{350}{25}$ 14	$\frac{400}{35}$ 11.4285	$\frac{450}{45}$ 10	$\frac{20}{5}$ 4	$\frac{70}{25}$ 2.8	$\frac{8}{3}$ 2.666	$\frac{5}{2}$ 2.5	$\frac{5}{2}$ 2.5
Orden final	1	2	3	4	5	6	7	8

La solución heurística es: $X^* = (1, 1, 0, 1, 1, 1, 1, 1)$ con $z(X^*) = 858$

Cuadro 4.11: *Iteraciones del algoritmo constructivo aplicado al problema de la mochila binario (4)*

# Iteración	b	Es ¿$\frac{b}{a_{i'}} < 1$? o ¿$\frac{b}{a_{i'}} \geq 1$?	Valor de x'_i
1	$b = 104$	$\frac{104}{25} > 1$	$x_1 = 1$
2	$b = 104 - 25 = 79$	$\frac{79}{35} > 1$	$x_2 = 1$
3	$b = 79 - 35 = 44$	$\frac{44}{245} < 1$	$x_3 = 0$
4	$b = 44 - 0 = 44$	$\frac{44}{5} > 1$	$x_4 = 1$
5	$b = 44 - 5 = 39$	$\frac{39}{25} > 1$	$x_5 = 1$
6	$b = 39 - 25 = 14$	$\frac{14}{3} > 1$	$x_6 = 1$
7	$b = 14 - 3 = 11$	$\frac{11}{2} > 1$	$x_7 = 1$
8	$b = 11 - 2 = 9$	$\frac{9}{2} > 1$	$x_7 = 1$

24 artículos
$maz\ z = 825,594x_1 + 1,677,009x_2 + 1,676,628x_3 + 1,523,970x_4 + 943,972x_5 + 97,426x_6 + 69,666x_7 + 1,296,457x_8 + 1,679,693x_9 + 1,902,996x_{10} + 1,844,992x_{11} + 1,049,289x_{12} + 1,252,836x_{13} + 1,319,836x_{14} + 953,277x_{15} + 2,067,538x_{16} + 675,367x_{17} + 853,655x_{18} + 1,826,027x_{19} + 65,731x_{20} + 901,489x_{21} + 577,243x_{22} + 466,257x_{23} + 369,261x_{24}$
sujeto a
$382,745x_1 + 799,601x_2 + 909,247x_3 + 729,069x_4 + 467,902x_5 + 44,328x_6 + 34,610x_7 + 698,150x_8 + 823,460x_9 + 903,959x_{10} + 853,665x_{11} + 551,830x_{12} + 610,856x_{13} + 670,702x_{14} + 488,960x_{15} + 951,111x_{16} + 323,046x_{17} + 446,298x_{18} + 931,161x_{19} + 31,385x_{20} + 496,951x_{21} + 264,724x_{22} + 224,916x_{23} + 169,684x_{24} \leq 6,404,180$
$x_i \in \{0, 1\}\ \forall i$
Óptimo $X = (1, 1, 0, 1, 1, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1)$
$z(X) = 13,549,094$

La solución heurística es:
$X = (1, 1, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 1)$ con $Z(X) = 13,415,886$

El correr el algoritmo en las 5 instancias anteriores, se obtienen los siguientes resultados:

4.5. Algoritmo heurístico: Búsqueda en su entorno

Para poder explicar los algoritmos heurísticos de búsqueda en su entorno es necesario saber qué es un vecino y una vecindad.

Definición 15 *Sean P un problema de optimización, X el espacio de soluciones de P y x una solución de P, $x' \in X$ es un vecino de x si se puede obtener directamente a partir de x mediante una operación llamada movimiento, esta puede ser una permutación o si el problema es binario puede ser intercambiar una entrada de 0 a 1 o de 1 a 0. [Papadimitriou (1998)]*

Ejemplo 16 *Consideremos la instancia del problema de la mochila con 8 artículos.*

Una solución generada de manera aleatoria es:

$$X_1 = (0, 0, 0, 0, 0, 0, 1, 0)$$

para obtener un vecino de X_1, se genera un número aleatorio $1 \leq r_1 \leq n$, supongamos que $r_1 = 4$, entonces el vecino es:

$$X_2 = (0, 0, 0, 1, 0, 0, 1, 0)$$

Si se desea obtener otro vecino, se genera otro número aleatorio $1 \leq r_2 \leq n$, supongamos que $r_1 = 1$, entonces el vecino es:

$$X_3 = (1, 0, 0, 0, 0, 0, 1, 0)$$

Ejemplo 17 *Si consideramos una instancia del problema del agente viajero con 5 ciudades.*
Una solución es:

$$X_1 = (1, 2, 3, 4, 5) \text{ (después de 5, regresamos a 1)}$$

Una forma de obtener un vecino es generar 2 números aleatorios $r_1 = 3$ y $r_2 = 5$, entonces, se intercambian las entradas de X_1, la solución vecina es:

$$X_2 = (1, 2, 5, 4, 3) \text{ (después de 3, regresamos a 1)}$$

Cabe mencionar que solo estamos considerando soluciones de las instancias, si contamos con los datos de cada una de ellas se podrá concluir si es factible o no, además de evaluar las soluciones en la función objetivo para escoger al mejor vecino.

Definición 16 *Sean P un problema de optimización, X el espacio de soluciones de P y x una solución de P, definimos $N(x) \subseteq X$ como una vecindad de x si y sólo si $\forall x' \in N(x)$, x' es vecino de x*

Observación 15 *El tamaño de la vecindad no es único, este se define al aplicar el algoritmo al problema a resolver.*

Observación 16 *Las vecindades en un problema combinatorio están definidas por las permutaciones, en ocasiones se recomienda solo hacer una permutación ya que si se generan más, da pie a que la vecindad crezca y el análisis nos lleve a una oscilación.*
Las vecindades de un problema continuo se definen como intervalos.

Procedimiento general: Los algoritmos de búsqueda en su entorno empiezan con una solución creada de manera aleatoria o utilizando un algoritmo constructivo, se genera un vecino o una vecindad, si este (o algún elemento de la vecindad) mejora el valor de la función objetivo, cambiamos de solución, si no mejora, se mantiene la solución actual; termina después de cierto número de iteraciones..

Observación 17 *Los algoritmos de búsqueda en su entorno se pueden utilizar en problemas de optimización continua y discreta, la desventaja de este método es que es muy probable de quedar estancado en un óptimo local, a pesar de que se inicia con una solución aleatoria.*

4.5.1. Algoritmo de búsqueda en su entorno aplicado al problema de la mochila

En esta sección aplicaremos el algoritmo a instancias binarias del problema de la mochila (las variables de decisión son 0 o 1)

Objetivo: Obtener una solución factible para una instancia del problema de la mochila.

1. Iniciar con una solución factible, esta puede ser generada de manera aleatoria. Sea S_0 dicha solución.

2. Genera una vecindad $N(S_0)$ en entorno a la solución actual S_0 (con ayuda de una operación se crean soluciones vecinas a S_0), éstas serán evaluadas en la función objetivo (z), aquí se debe penalizar a las soluciones que no sean factibles.

3. Se escoge la mejor solución de la vecindad, sea $S_1 \in N(S_0)$. Si $z(S_1) > z(S_0)$ (en el caso de maximizar), entonces, $S_1 = S_0$, en caso contrario S_0 no cambia, ir a 4.

4. Si no se cumple el criterio de parada ir a 2, en caso contrario, se encontró la mejor solución, ésta es S_0 y $z(S_0)$ es el mejor valor de la función objetivo.

Los algoritmos de búsqueda pueden generar soluciones muy buenas, esto depende del criterio de parada y cómo se generan las vecindades, que con ayuda de la estructura del problema hará que sea eficiente el algoritmo.
En el siguiente diagrama se resumen los pasos del algoritmo de búsqueda en su entorno:

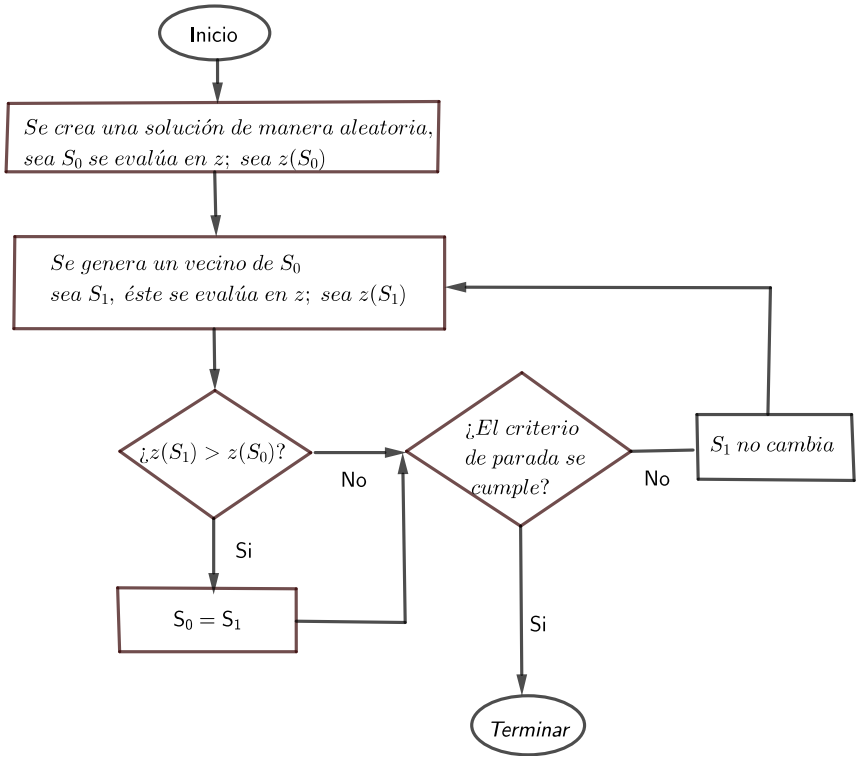


Figura 4.2: *Heurístico búsqueda en su entorno para el problema binario*

Fuente: Elaboración propia utilizando Geogebra 5 (2016)

Antes de aplicar el algoritmo se especificarán ciertos puntos importantes.

- **Solución inicial:** La solución se generará de manera aleatoria, realizando los siguientes pasos:

  1. $i = 0$ contador de iteraciones (inicialización)
  2. Generar un número aleatorio $a_1 \in [0, 1]$. Si $0 < a_1 \leq 0.5 \rightarrow x_i = 0$, en otro caso $x_i = 1$
  3. $i = i + 1$
  4. Si $i \leq n$ ir a 2, en otro caso, terminar, la solución inicial es $X = (x_1, x_2, \ldots, x_n)$

- **Vecinos:** Los vecinos se generan de la siguiente manera, sea X una solución al problema, esta tiene n entradas; se genera un número aleatorio $a_2 \in [1, n]$ se intercambia la entrada x_{a_2}

  a) Si $x_{a_2} = 0$ se cambia por $x_{a_2} = 1$
  b) Si $x_{a_2} = 1$ se cambia por $x_{a_2} = 0$

  Las demás entradas conservan su valor.

- **Solución no factible:** Una de las desventajas de los algoritmos heurísticos es que en las iteraciones podemos encontrar soluciones no factibles y estas se deben penalizar en la evaluación de la función objetivo con el fin de no tomarlas en cuenta. (La penalización debe ser tan grande como sea necesario para que la solución no sea atractiva). En nuestro caso la penalización será 500 para las instancias 1, 2, 3 y 4, en la instancia 5 será de $5,000,000$; en cada problema la penalización cambia, dependiendo de los coeficientes de la función objetivo.

Se aplicará este algoritmo a la instancia con $n = 8$, los datos son:

$$c = [350, 400, 450, 20, 70, 8, 5, 5]$$

$$a = [25, 35, 45, 5, 25, 3, 2, 2]$$

$$b = 104$$

El modelo matemático correspondiente a esta instancia es:

$$maz\ z = 350x_1 + 400x_2 + 450x_3 + 20x_4 + 70x_5 + 8x_6 + 5x_7 + 5x_8$$
$$sujeto\,a$$
$$25x_1 + 35x_2 + 45x_3 + 5x_4 + 25x_5 + 3x_6 + 2x_7 + 2x_8 \leq 104$$
$$x_i \in \{0, 1\}\ \forall i$$

La solución inicial obtenida de manera aleatoria es:

$$X_{inicial} = (1, 0, 0, 1, 1, 0, 0, 0)$$

$$F.\ objetivo_{inicial} = 440.0$$

Se realizaron 10 iteraciones, la información de cada iteración estará contenida en la siguiente tabla:

Cuadro 4.12: *Resumen del algoritmo búsqueda en su entorno (instancia 8 artículos)*

# aleatorio (a_2)	*Vecino*	*¿Es factible?*	$Z(Vecino)$	*¿Cambia la solución?*
1	$X_1 = (0,0,0,1,1,0,0,0)$	Si	$Z(X_1) = 90$	no
3	$X_2 = (1,0,1,1,1,0,0,0)$	Si	$Z(X_2) = 890$	si
8	$X_3 = (1,0,1,1,1,0,0,1)$	Si	$Z(X_3) = 895$	si
2	$X_4 = (1,1,1,1,1,0,0,1)$	No	$Z(X_4) = 1295 - 500 = 795$	no
7	$X_5 = (1,0,1,1,1,0,1,1)$	Si	$Z(X_5) = 900$	si
1	$X_6 = (0,0,1,1,1,0,1,1)$	Si	$Z(X_6) = 550$	no
3	$X_7 = (1,0,0,1,1,0,1,1)$	Si	$Z(X_7) = 450$	no
6	$X_8 = (1,0,1,1,1,1,1,1)$	No	$Z(X_8) = 908 - 500 = 408$	no
6	$X_9 = (1,0,1,1,1,1,1,1)$	No	$Z(X_9) = 908 - 500 = 408$	no
3	$X_{10} = (1,0,0,1,1,0,1,1)$	Si	$Z(X_{10}) = 450$	no

Nuestro criterio de parada es a las 10 iteraciones, la solución óptima es:

$$X_5 = (1,0,1,1,1,0,1,1) \; y \; Z(X^*) = 900$$

que coincide con la solución óptima de la instancia.
Esta solución se obtuvo en la iteración 5, pero como se mencionó anteriormente, los algoritmos son ciegos y no es posible identificar cuando se ha llegado al óptimo.
Con lo anterior podemos concluir que con la solución inicial $X_{inicial} = (1,0,0,1,1,0,0,0)$, la efectividad es del 100 %, esto no siempre ocurre ya que si procedemos de la misma manera (construir la solución inicial de manera aleatoria) no siempre se llegará al óptimo, esto se verá más adelante.
Para poder hacer un análisis con respecto a la efectividad del algoritmo se aplicará este a las 5 instancias anteriores con los siguientes condiciones:

- Por cada instancia se correrá el algoritmo con 50, 100, 500,y 1000 iteraciones,
- Por cada instancia y evento se correrán 50 veces.

La información de la tabla es la siguiente:
Número de iteraciones y de las iteraciones que se realizaron, ¿cuántas cayeron en el óptimo?

Esta se proporciona en los siguientes cuadros.

Cuadro 4.13: *Búsqueda en su entorno (instancia 1)*

N^o de iteraciones	N^o veces que se obtuvo el óptimo
50	11
100	21
500	15
1000	15

Cuadro 4.14: *Búsqueda en su entorno (instancia 2)*

N^o de iteraciones	N^o veces que se obtuvo el óptimo
50	6
100	2
500	4
1000	6

Cuadro 4.15: *Búsqueda en su entorno (instancia 3)*

N^o de iteraciones	N^o veces que se obtuvo el óptimo
50	0
100	2
500	2
1000	0

Cuadro 4.16: *Búsqueda en su entorno (instancia 4)*

N^o de iteraciones	N^o veces que se obtuvo el óptimo
50	3
100	14
500	5
1000	3

Instancia 5

Para la instancia no se pudo llegar al óptimo, el mejor valor obtenido es: $13,267,280$
La efectividad del algoritmo es:

$$\%\, efectividad = 100 - \left[\left(\frac{13,549,094 - 13,267,280}{13,549,094}\right) \times 100\right] = 97.92\,\%$$

Con los resultados obtenidos podemos concluir que el algoritmo es aceptable ya que a pesar de que no siempre nos proporciona el óptimo, en alguna(s) iteración

(iteraciones) si la genera, cabe señalar la importancia que tienen ciertos criterios para poder tener estos resultados, estos son:

- La penalización, ésta debe ser los suficientemente grande para no caer en soluciones no factibles, ya que si en alguna iteración S_0 es una solución no factible nunca saldríamos de ella puesto que ninguna solución factible la mejoraría y en consecuencia el óptimo será una solución no factible.

- El número de iteraciones no es un factor que influya en el resultado final.

- La solución inicial es muy importante, se recomienda generarla de manera aleatoria, ya que si utilizamos por ejemplo el algoritmo glotón para generarla nos estancamos en el óptimo local.
  Supongamos que se utiliza como solución inicial la obtenida con el algoritmo constructivo, esta es: $S_0 = (1, 1, 0, 1, 1, 1, 1, 1)$ con $z(S_0) = 858$
  Los vecinos que se pueden generar a partir de S_0 (con sus respectivas funciones objetivo) son:

Cuadro 4.17: *Vecinos de* $S_0 = (1, 1, 0, 1, 1, 1, 1, 1)$

Vecino	Función objetivo
$X_1 = (0, 1, 0, 1, 1, 1, 1, 1)$	508
$X_2 = 1, 0, 0, 1, 1, 1, 1, 1)$	458
$X_3 = (1, 1, 1, 1, 1, 1, 1, 1)$	1308
$X_4 = 1, 1, 0, 0, 1, 1, 1, 1)$	838
$X_5 = (1, 1, 0, 1, 0, 1, 1, 1)$	788
$X_6 = 1, 1, 0, 1, 1, 0, 1, 1)$	850
$X_7(1, 1, 0, 1, 1, 1, 0, 1)$	853
$X_8 = 1, 1, 0, 1, 1, 1, 1, 0)$	853

  Observamos que la única solución que mejora el valor de z es X_3, pero no es factible, si llegamos a cambiar a S_0 por X_3 nunca saldríamos de ella ya que la solución óptima es $X^* = (1, 0, 1, 1, 1, 0, 1, 1)$ con $z(X^*) = 900$ y para llegar a ella a partir de X_3 necesitamos de dos movimientos, esto no es posible, ya que no sería un vecino; no saldríamos de la solución no factible sin importar cuantas iteraciones se realicen ya que la función objetivo no mejoraría a la actual (que no es factible).
  Por esto se recomienda generar a la solución inicial de forma aleatoria.

4.6. Recocido Simulado (Metaheurístico)

[Dowsland, K. (2003)]. El método de Recocido simulado tiene sus orígenes en 1983, en el artículo: Kirkpatrick, S., Gelatt Jr, C. D., Vecchi, M. P. Optimization by simulated annealing. science, 220(4598), 671-680.
El nombre de recocido simulado es debido a su analogía con el proceso del recocido

físico con sólidos, en el cual un sólido cristalino se calienta y luego se deja enfriar lentamente hasta que alcanza su configuración de red cristalina más regular posible y por lo tanto está libre de defectos.
Para ilustrar esto, se comparan dos técnicas de enfriamiento, el recocido y el templado.

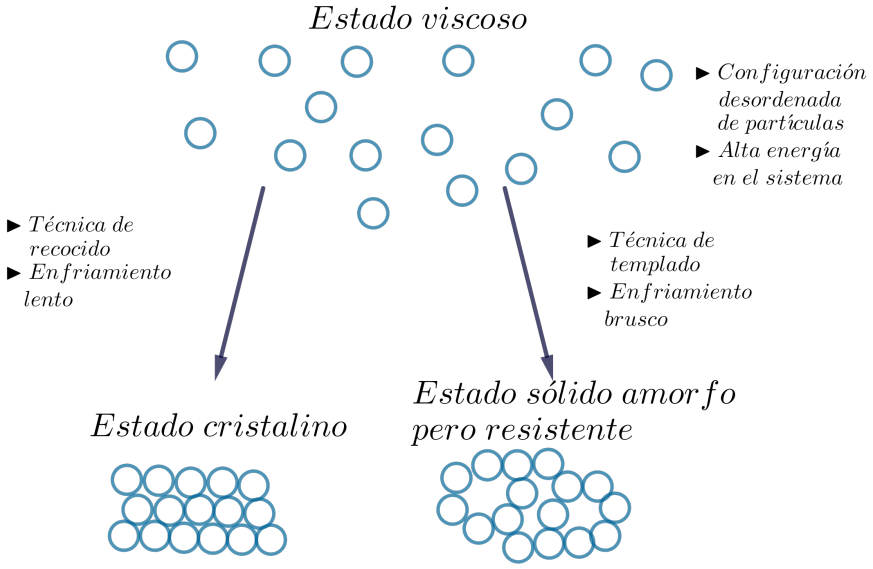


Figura 4.3: *Estados del vidrio*

Fuente: Elaboración propia utilizando Geogebra 5 (2016)

Un antecedente del recocido simulado es el algoritmo de Metropolis, en termodinámica estadística funciona de la siguiente manera, genera una perturbación aleatoria en el sistema (baja la temperatura), se calculan los cambios de energía resultantes y si hay una caída energética, el cambio se acepta, en otro caso se acepta con una probabilidad

$$P(\Delta f, T) = e^{\frac{-\Delta f}{kT}} \text{——————————} (1)$$

Donde k es la constante de Boltzmann, dicha expresión se obtiene de las leyes de la termodinámica que dicen que a una temperatura T la probabilidad de un incremento energético de magnitud Δf está dada por: (1)
Es posible asociar cada uno de los elementos de un problema de optimización con los elementos del sistema de enfriamiento del metal, esto es:

Cuadro 4.18: *Elementos de un problema de optimización*

Simulación de un sistema termodinámico	**Problema de optimización**
Estado	Alternativa, es decir, solución factible
Cambio Energético	Cambio en la función objetivo
Cambio de estado (enfriamiento)	Solución Búsqueda local
Temperatura	Parámetro de control (T)
Estado estable (reposo)	Solución heurística

Como la constante de Boltzmann no tiene significado para los problemas de optimización combinatoria, la probabilidad de escoger una solución peor está dada por la siguiente expresión:

$$P(\Delta f, T) = e^{\frac{-\Delta f}{T}}$$

Procedimiento general
Al aplicar el algoritmo de recocido simulado a un problema de optimización, en cada iteración se comparan los valores de las funciones objetivo de dos soluciones (la solución actual y una solución recién seleccionada) y se tienen dos casos:

- Las soluciones que mejoran el valor de la función objetivo siempre se aceptan.
- Si una solución no mejora el valor de la función objetivo se acepta con la esperanza de escapar de los óptimos locales en busca de óptimos globales. La probabilidad de aceptar soluciones que no mejoran depende de un parámetro de temperatura, que generalmente disminuye lentamente con cada iteración del algoritmo.

Una característica clave de este algoritmo es que proporciona un medio para escapar de los óptimos locales, permitiendo movimientos de descenso (es decir, movimientos que empeoran el valor de la función objetivo). A medida que el parámetro de temperatura se reduce a cero, los movimientos de descenso ocurren con menos frecuencia. Propiamente el recocido del acero consiste en calentar el metal y bajar la temperatura paulatinamente para poder amoldarlo hasta que llegue a una fase de estabilidad, si el proceso de enfriamiento es muy rápido se obtienen sólidos no cristalinos, en otras palabras, el calor causa que los átomos aumenten su energía y que puedan así desplazarse de sus posiciones iniciales (un mínimo local de energía); el enfriamiento lento les da mayores probabilidades de recristalizar en configuraciones con menor energía que la inicial (mínimo global).
El método de recocido simulado puede ser utilizado en problemas de optimización continua y discreta, se tiene que tener mucho cuidado en buscar una buena transformación para bajar la temperatura, ya que el bajarla de manera rápida aumenta la probabilidad de caer en un óptimo local.

4.6.1. Recocido simulado para el problema de la mochila

Utilizaremos la siguiente notación para explicar el algoritmo de recocido simulado.
Sean:

- P el problema de optimización a resolver.
- $z(X)$ la función a optimizar.
- S_0 una solución inicial.
- $N(S_0)$ la vecindad generada en torno a S_0
- T: Temperatura, es el parámetro para aceptar una mala solución. Se recomienda empezar con una temperatura alta para que permita movimientos erráticos al principio del proceso. Se pueden hacer varias pruebas con diferentes temperaturas hasta que el resultado sea el que necesitamos, salir de los óptimos locales.
- $g(T)$: velocidad de enfriamiento, ésta puede ser aritmética, geométrica, etc. Muchos estudios teóricos han descubierto que si la temperatura decrece suficientemente lento, el proceso converge a la solución óptima, aunque esto también depende de la estructura de cada problema.
- n número de iteraciones, criterio de parada. (en ocasiones también el criterio de parada puede ser una cota para $z(X)$, o el decremento de temperatura,)
- p es un número aleatorio ($p \in [0, 1]$), este sirve para aceptar o no una solución que no mejora el valor de z, si $p \leq P(\Delta z, T) = e^{\frac{-\Delta z}{T}}$, entonces, la solución es aceptada.

Objetivo: Obtener una solución factible de una instancia del problema de la mochila.

1. Empezar con cierta solución inicial S_0 que puede ser generada de forma aleatoria o utilizando búsqueda en su entorno o algún otro heurístico.
2. Mediante un proceso se genera la vecindad $N(S_0)$ en torno a S_0 y se evalúan dichos vecinos en $z(X)$ penalizando a las soluciones no factibles, ir a 3
3. Para un problema de maximización:
   - Caso 1: Sea $S^* = max\{z(S)|S \in N(S_0)\}$ si $z(S^*) > z(S_0)$, entonces la solución es aceptada, es decir, $S_0 = S^*$
   - Caso 2: Sea $S^* = max\{z(S)|S \in N(S_0)\}$ si $z(S^*) < z(S_0)$, entonces, se calcula lo siguiente:
     - se genera un número aleatorio $p \in [0, 1]$

- $\Delta z = z(S_0) - z(S*)$
- $e^{\frac{-\Delta z}{T}}$

Si $p < e^{\frac{-\Delta z}{T}}$ S^* se acepta $S_0 = S^*$, en caso contrario se rechaza $S^* = S^*$.

4. Bajar la temperatura, $g(T) = T * 0.9$.
   Esto hace que la probabilidad de aceptar una solución peor sea menor con forme pasan las iteraciones.

5. Si el criterio de parada no se ha cumplido, ir a 2. En otro caso terminar, la solución es la mejor obtenida hasta este momento.

En el siguiente diagrama de flujo se resumen los pasos a seguir para aplicar recocido simulado al problema de la mochila binario:

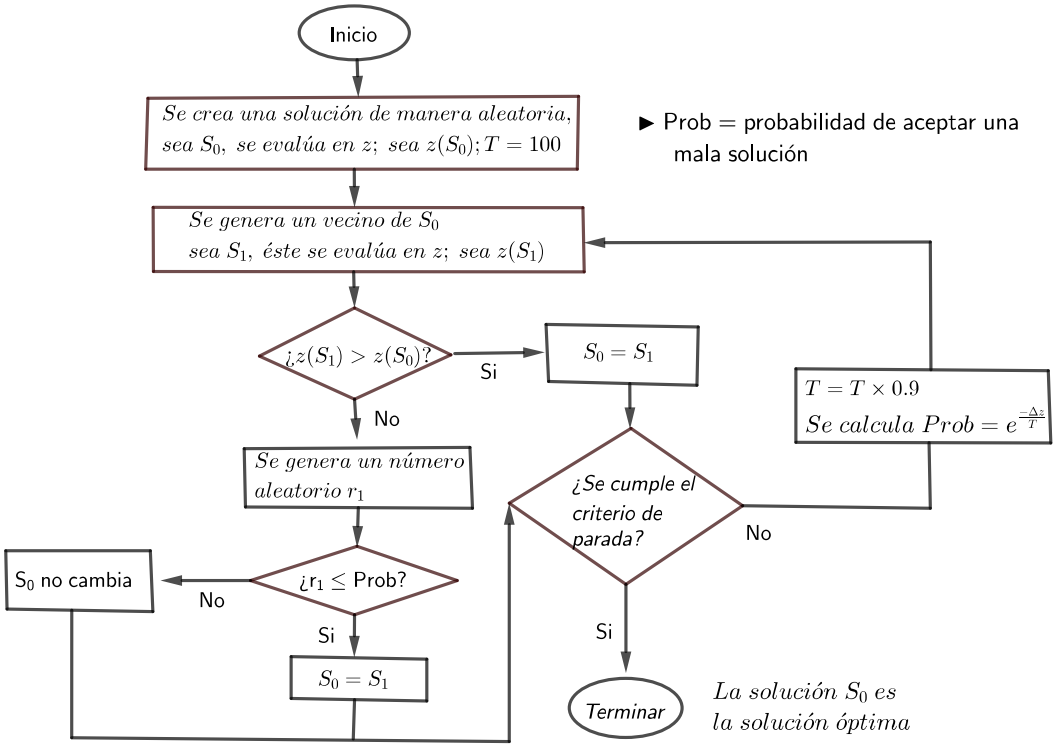


Figura 4.4: *Recocido simulado para el problema binario*

Fuente: Elaboración propia utilizando Geogebra 5 (2016)

Implementación
Antes de aplicar el algoritmo se especificarán ciertos puntos importantes.

- La temperatura inicial será $T = 100$

- La transformación para hacer decrecer la temperatura es: $T = T \times 0.9$
- **Solución inicial:** La solución se generará de manera aleatoria, realizando los siguientes pasos:

  1. $i = 0$ contador de iteraciones (inicialización)
  2. Generar un número aleatorio $a_1 \in [0, 1]$. Si $0 < a_1 \leq 0.5 \rightarrow x_i = 0$, en otro caso $x_i = 1$
  3. $i = i + 1$
  4. Si $i < n$ ir a 2, en otro caso, terminar, la solución inicial es $X = (x_1, x_2, \ldots, x_n)$

- **Vecinos:** Los vecinos se generan de la siguiente manera, sea X una solución al problema, esta tiene n entradas; se genera un número aleatorio $a_2 \in [1, n]$ se intercambia la entrada x_{a_2}

  a) Si $x_{a_2} = 0$ se cambia por $x_{a_2} = 1$

  b) Si $x_{a_2} = 1$ se cambia por $x_{a_2} = 0$

  Las demás entradas conservan su valor.

- **Solución no factible:** La penalización de una solución no factible es 500

Aplicaremos el algoritmo en la instancia de 8 artículos:

$$maz\ z = 350x_1 + 400x_2 + 450x_3 + 20x_4 + 70x_5 + 8x_6 + 5x_7 + 5x_8$$
$$sujeto\,a$$
$$25x_1 + 35x_2 + 45x_3 + 5x_4 + 25x_5 + 3x_6 + 2x_7 + 2x_8 \leq 104$$
$$x_i \in \{0, 1\}\ \forall i$$

En el algoritmo se aplicaran 6 iteraciones y ese será nuestro criterio de parada.
La solución inicial obtenida de manera aleatoria es:

$$X_{inicial} = (1, 0, 0, 0, 0, 0, 1, 1)$$

$$F.\ objetivo_{inicial} = 360.0$$

$$T = 100$$

- **Iteración 1:** Se genera un número aleatorio $a_2 = 5$ para crear un vecino, esto es, en la solución inicial se cambia la entrada 5 por un 1, quedando: $X_1 = (1, 0, 0, 0, 1, 0, 1, 1)$, al evaluar en $z(X_1) = 430$.
  Como mejora el valor de z, entonces cambiamos de solución.

$$T = 90$$

- **Iteración 2:** Se genera un número aleatorio $a_2 = 4$ para crear un vecino, esto es, en la solución X_1 se cambia la entrada 4 por un 1, quedando: $X_2 = (1, 0, 0, 1, 1, 0, 1, 1)$, al evaluar en $z(X_2) = 450$.
  Como mejora el valor de z, entonces cambiamos de solución.

$$T = 81$$

- **Iteración 3:** Se genera un número aleatorio $a_2 = 7$ para crear un vecino, esto es, en la solución X_2 se cambia la entrada 7 por un 0, quedando: $X_3 = (1, 0, 0, 1, 1, 0, 0, 1)$, se evalúa en z y se obtiene $z(X_3) = 445$
  Como no mejora el valor de z, entonces, se genera otro número aleatorio $p = 0.8603$, se evalúa en $P(\Delta f, T) = e^{\frac{-\Delta f}{T}}$

$$\Delta f = 450 - 445 = 5$$

$$P(\Delta f, T) = e^{\frac{-5}{81}} = 0.9401$$

  Como $0.8603 < 0.9401$ se acepta la solución y la cambiamos.

$$T = 72.9$$

  Las iteraciones se pueden resumir en el siguiente cuadro:

La información de las iteraciones anteriores se resume en la siguiente tabla:

Cuadro 4.19: *Recocido simulado, iteraciones 1,2 y 3*

# aleat. (a_2)	*Vecino*	$Z(Vecino)$	*¿Mejora la solución?*	Tem.	*# aleat.* (p)	*Prob*
5	$X_1 = (1, 0, 0, 0, 1, 0, 1, 1)$	$Z(X_1) = 430$	Si	90	$--$	$--$
4	$X_2 = (1, 0, 0, 1, 1, 0, 1, 1)$	$Z(X_2) = 450$	Si	81	$--$	$--$
7	$X_3 = (1, 0, 0, 1, 1, 0, 0, 1)$	$Z(X_3) = 445$	No	72.9	0.8603	0.9401

Se realizaron 6 iteraciones, la información de las siguientes 3 iteraciones está contenida en la siguiente tabla:

Cuadro 4.20: *Recocido simulado, iteraciones 4,5 y 6*

# aleatorio (a_2)	*Vecino*	$Z(Vecino)$	*¿Mejora la solución?*	Tem.	*# aleatorio* (p)	*Prob*
3	$X_4 = (1, 0, 1, 1, 1, 0, 0, 1)$	$Z(X_4) = 895$	Si	65.61	$--$	$--$
7	$X_5 = (1, 0, 1, 1, 1, 0, 1, 1)$	$Z(X_5) = 900$	Si	59.0490	$--$	$--$
8	$X_6 = (1, 0, 1, 1, 1, 0, 1, 0)$	$Z(X_6) = 895$	No	53.1441	0.8626	0.1523

Como la solución obtenida en la iteración 6 no mejora a la z anterior, preguntamos, ¿$p < P(\Delta f, T) = e^{\frac{-\Delta f}{T}}$?, como no se cumple, la solución no se acepta; llegamos a la iteración 6, terminamos, la mejor solución encontrada hasta este momento es $X_5 = (1, 0, 1, 1, 1, 0, 1, 1)$ con $z(X^*) = 900$ que en este caso es la solución óptima.
Podemos notar que la temperatura en cada iteración decrece lentamente y la probabilidad para aceptar una solución no atractiva también decrece, aunque de manera exponencial, esto hace que conforme realizamos más iteraciones sea menos probable escoger una solución que no mejora a la solución anterior.
Aplicamos el algoritmo de recocido simulado en las 5 instancias anteriores (con las mismas condiciones que el algoritmo de búsqueda en su entorno) y obtenemos los siguientes resultados:

Cuadro 4.21: *Recocido simulado, instancia 1*

N^o de iteraciones	N^ode veces que cae en el óptimo después de 50 corridas
50	19
100	19
500	13
1000	19

Cuadro 4.22: *Recocido simulado, instancia 2*

N^o de iteraciones	N^ode veces que cae en el óptimo después de 50 corridas
50	13
100	7
500	11
1000	8

Cuadro 4.23: *Recocido simulado, instancia 3*

N^o de iteraciones	N^ode veces que cae en el óptimo después de 50 corridas
50	0
100	1
500	4
1000	1

Cuadro 4.24: *Recocido simulado, instancia 4*

N^o de iteraciones	N^ode veces que cae en el óptimo después de 50 corridas
50	3
100	11
500	6
1000	11

Cuadro 4.25: *Recocido simulado, instancia 5*

N^o de iteraciones	Mejor valor de z después de 50 corridas
50	$13,316,306$
100	$13,141,728$
500	$13,271,922$
1000	$13,383,358$

La mejor solución encontrada con este algoritmo es: $13,383,358$, tomando en cuenta este valor, la efectividad del algoritmo es:

$$\%\, efectividad = 100 - \left[\left(\frac{13,549,094 - 13,383,358}{13,549,094}\right) \times 100\right] = 98.7767\,\%$$

Con los resultados anteriores podemos concluir que el algoritmo propuesto en esta sección (con los criterios y parámetros descritos) es aceptable y hasta ahora es el mejor de los tres algoritmos que hemos desarrollado, cabe mencionar que este algoritmo puede modificarse con alguno de los siguientes cambios:

- Manera de generar a los vecinos.
- Transformación para bajar la temperatura.
- Probabilidad de aceptar una mala solución.
- Generar una solución inicial.

En el artículo de Moradi, N., Kayvanfar, V., & Rafiee, M. (2021) proponen 3 modificaciones al algoritmo propuesto anteriormente, estos son:

1. Comparan 2 maneras diferentes para obtener la solución inicial, estas son: Aleatoriamente y mediante un algoritmo glotón; como conclusión mencionan que si utilizan un algoritmo constructivo voraz (glotón) para crear la solución inicial, al final del algoritmo se llegaría a la solución óptima con una probabilidad alta.

2. Proponen un procedimiento de transformación para una solución no factible y convertirla en factible; este procedimiento ordena los cocientes $\frac{c_i}{a_i}$ de menor a mayor y saca de la mochila al primer artículo del que se tiene menor beneficio marginal, si la solución sigue siendo no factible pasa al siguiente articulo con menos beneficio marginal, este procedimiento se repite hasta que se tenga una solución factible.

3. Un criterio que incorporan al algoritmo es que la solución obtenida sature la capacidad de la mochila.

4. Por último agregan el enfoque poblacional al algoritmo, es decir, utilizan el cruzamiento y mutación para generar vecinos.

4.7. Búsqueda Tabú (Metaheurístico)

El término Tabú (taboo) proviene de la Polinesia, donde es usado por los aborígenes de la isla Tonga para referirse a cosas que no deben ser tocadas ya que son sagradas.
Búsqueda Tabú se basa en la siguiente metodología:

Es mejor una mala decisión basada en información, que una buena decisión al azar, ya que en un sistema que emplea memoria, una mala elección basada en una estrategia proporcionará claves útiles para continuar la búsqueda. Una buena elección fruto del azar no proporcionará ninguna información para posteriores acciones.

Algunas características de este método es el uso de memoria adaptativa y que además explota patrones históricos de la búsqueda, es decir, se guía por la historia, de manera opuesta a los procesos que confían casi exclusivamente en la aleatorización.

El nombre y la metodología se deben a Fred Glover, en 1986 en el artículo *Future paths for integer programming and links to artificial intelligence* define a búsqueda tabú como una guía para que los algoritmos de búsqueda local puedan explorar el espacio de soluciones y lleguen más allá del óptimo local.

Toma de la inteligencia artificial el concepto de memoria y lo implementa mediante estructuras simples con el objetivo de dirigir la búsqueda teniendo en cuenta la historia de ésta, es decir, el procedimiento trata de extraer información de lo sucedido y actuar en consecuencia. En este sentido puede decirse que hay un cierto aprendizaje y que la búsqueda es inteligente.
La estructura de memoria de la búsqueda tabú funciona mediante cuatro dimensiones principales, estas son:

- Reciente
- Frecuencia
- Calidad
- Influencia.

Estas características junto con antecedentes de conectividad y estructuras lógicas hacen posible la definición de dos tipos de memorias:

- Memoria a corto plazo para evitar la reversión de movimientos recientes.
- Memoria a largo plazo (memoria de frecuencia) para reforzar componentes atractivos.

También es importante señalar que Glover no vio la búsqueda tabú como una heurística más, sino más bien como una metaheurística, es decir, una estrategia general para guiar y controlar las heurísticas de búsqueda para adaptarlas a los problemas en cuestión y obtener mejores resultados.
Procedimiento general [Glover, F.(2003)].
Como es un algoritmo de búsqueda, empezamos con una solución inicial, sea S_0, a partir de esta se genera una vecindad $N(S_0)$, escogemos a la mejor solución de $N(S_0)$, sea X^* (tomando en cuenta penalizaciones por la no factibilidad o por la tabla de frecuencia) y hacemos $S_0 = X^*$ siempre y cuando X^* no esté en la lista tabú, el criterio de parada puede ser el número de iteraciones.
Ahora explicaremos en qué consisten la lista tabú y la tabla de frecuencia.

- Lista tabú: Contiene a las soluciones que se han visitado recientemente, es decir, supongamos que X_1 era S_0 en la iteración 3 y en la iteración 4 cambio, S_0 ahora es X_7, entonces X_1 entra a la lista tabú y estará prohibido este movimiento en determinado número de iteraciones (esto varia según el problema a resolver), el número de iteraciones que estará prohibido disminuye una unidad por cada iteración que se realiza.

  Existe un criterio llamado criterio de aspiración, éste se aplica a la lista tabú si se cae en la siguiente situación: Supongamos que estamos en la iteración j, se genera $N(S_0)$ y al evaluar dichas soluciones en z, la mejor es X_i que está en la lista tabú, entonces, es posible hacer el movimiento si X_i "No es tan tabú", es decir, que ya esté próxima a salir, este criterio se fijará de manera diferente para cada problema.
  El criterio de aspiración también puede ser por atributo, es decir el valor de la función objetivo.

  La lista tabú es uno de los elementos que difiere de la búsqueda local, esta se utiliza para:

  - Evitar caer en ciclos, ya que coloca en la lista a las soluciones previamente visitadas y las "prohibe"durante algunas iteraciones para no regresar a ellas.
  - Almacena en una memoria a corto plazo de la búsqueda a ciertas soluciones (la lista tabú) y generalmente solo se registra una cantidad de información fija y bastante limitada.

- Tabla de frecuencia: Almacena en una memoria a largo plazo de la búsqueda la frecuencia de las soluciones que se han visitado, es decir, se contabiliza el

número de veces que se ha caído en una solución, después de determinado número de repeticiones se puede penalizar dicha solución para evitar caer en un ciclo, esto es para hacerla menos atractiva ya que puede ser un óptimo local.

En el siguiente diagrama de flujo se explican los pasos a seguir para aplicar búsqueda tabú en un problema binario

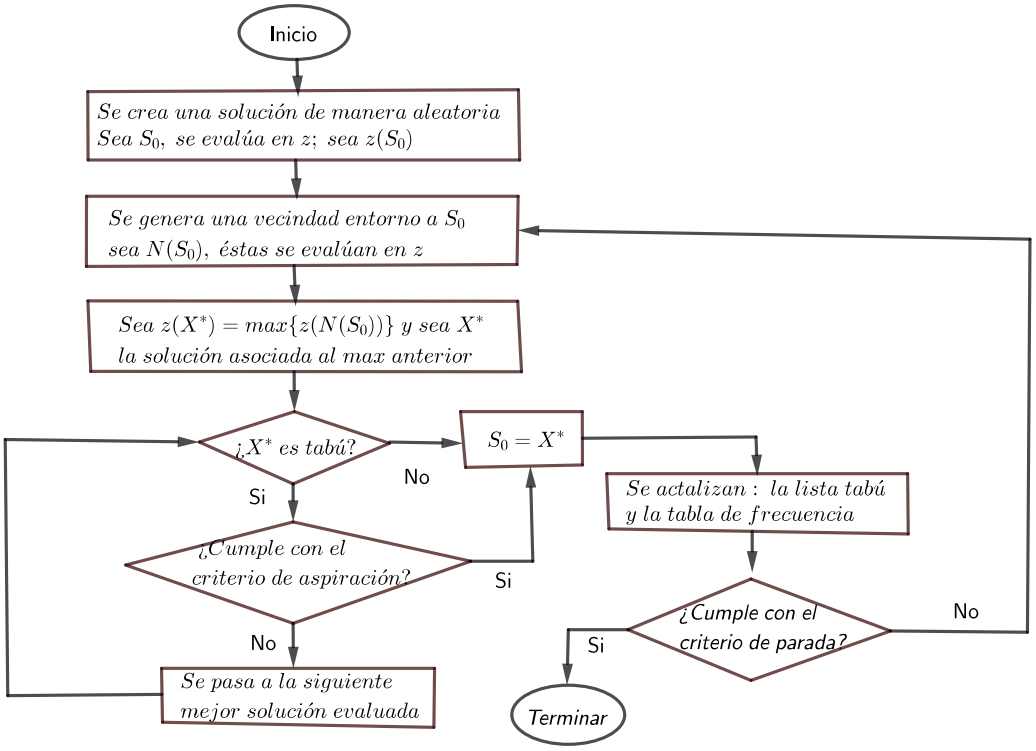


Figura 4.5: *Búsqueda Tabú para el problema binario*

Fuente: Elaboración propia utilizando Geogebra 5 (2016)

4.7.1. Búsqueda tabú aplicado al problema de la mochila

Objetivo: Obtener una solución factible para una instancia del problema de la mochila.

1. Inicialización
   Elegir (construir) una solución inicial S_0 de manera aleatoria, evaluarla en z, sea $z(S_0)$
   T es la lista tabú, en la primer iteración $T = \emptyset$

2. Construir una vecindad en torno a S_0, sea $N(S_0)$

3. Evaluar en z $\forall X_i \in N(S_0)$, se debe tomar en cuenta las penalizaciones por la no factibilidad o por la tabla de frecuencia.

4. Obtener $X^* = max\{z(X_i)|X_i \in N(S_0)\}$

5. Si $X^* \in T$ nos preguntamos, ¿Se cumple con el criterio de aspiración?

   - Si la respuesta es si, entonces, $S_0 = X^*$, ir a 6
   - Si la respuesta es no, quitamos a X^* de $N(S_0)$, ir a 4

6. ¿Se cumple el criterio de parada?

   - Si la respuesta es si, entonces, terminamos S_0 es la solución óptima y $z(S_0)$ el valor óptimo.
   - Si la respuesta es no, actualizamos la lista tabú, se agrega el movimiento reciente a la lista tabú y decrece en 1 la prohibición de las soluciones en la lista tabú. ir a 2

   El criterio de parada puede ser

   - Después de un número fijo de iteraciones.
   - Después de un número de iteraciones posteriores a una mejora en la función del objetivo.
   - Cuando la función objetivo alcanza un valor pre-especificado.

Ejemplo 18 *Resolveremos el siguiente problema de la mochila utilizando búsqueda tabú.*

$$\begin{gathered} max\ z = 70x_1 + 20x_2 + 39x_3 + 37x_4 + 7x_5 + 5x_6 + 10x_7 \\ s.a.\ 31x_1 + 10x_2 + 20x_3 + 19x_4 + 4x_5 + 3x_6 + 6x_7 \leq 50 \\ x_i \in \{0,1\}\ para\ i = 1, ...7 \end{gathered}$$

Primero daremos algunas especificaciones acerca de la tabla y criterios que se utilizarán.

- *Se aplicará el algoritmo en* 10 *iteraciones, este será el criterio de parada.*
- *Los vecinos se construirán al intercambiar una entrada con cero por uno o viceversa de manera aleatoria.*
- *La lista tabú estará formada por el movimiento correspondiente a la mejor solución en esa iteración. El número de iteraciones que permanecerá en la lista es* 3
- *No utilizaremos el criterio de aspiración*

- *Si la solución X_r es no factible se penalizará en la función objetivo $f(X_r) = f(X_r) - 100$*
- *El primer renglón de la tabla contiene: Las primeras 7 columnas son las variables del problema, la columna 8 es la evaluación de la función objetivo, incluyendo las penalizaciones por la no factibilidad , la última columna es la suma de las a_1 cuya $x_i = 1$, esto nos ayuda a identificar si la solución es factible o no.*
- *La solución actual (en cada iteración) se encontrará en el segundo renglón de la tabla.*
- *A partir del tercer renglón son los vecinos, en nuestro caso se generarán 5 vecinos en cada iteración.*
- *Marcaremos la mejor solución en cada iteración con un asterisco.*

Se construye una solución inicial aleatoriamente, esta es:

$$S_0 = (0, 0, 1, 0, 0, 0, 1)$$

con

$$Z(S_0) = 49 \; \Sigma a_i = 26$$

Cuadro 4.26: *Búsqueda tabú, iteración 1*

1	2	3	4	5	6	7	F. Obj.	Σa_i
0	0	1	0	0	0	1	49	26
0	0	1	0	1	0	1	61	30
0	0	1	0	0	1	1	54	29
0	0	1	0	0	0	0	39	20
0	0	0	0	0	0	1	10	6
0	1	1	0	0	0	1	69*	36

Lista tabú

1	2	3	4	5	6	7
0	3	0	0	0	0	0

Frecuencia

Solución	Frecuencia
0, 1, 1, 0, 0, 0, 1	1

El movimiento que se hizo fue el de la variable x_2, cambio de ser 0 a 1, por lo que este movimiento será prohibido durante tres iteraciones (la variable x_2 no puede cambiar de valor durante tres iteraciones.) La nueva solución es: $S_0 = (0, 1, 1, 0, 0, 0, 1)$ con $z(S_0) =$ 69

Cuadro 4.27: *Búsqueda tabú, iteración 2*

1	2	3	4	5	6	7	F. Obj.	Σa_i
0	1	1	0	0	0	1	69	36
0	1	0	0	0	0	1	30	16
0	1	1	0	0	1	1	74*	39
0	1	1	0	0	0	0	59	30
1	1	1	0	0	0	1	69	36
0	1	0	0	0	0	1	30	16

Lista tabú

1	2	3	4	5	6	7
0	2	0	0	0	3	0

Frecuencia

Solución	Frecuencia
$0, 1, 1, 0, 0, 0, 1$	1
$0, 1, 1, 0, 0, 1, 1$	1

Entra a la lista tabú la variable x_6 (variable en la que se hizo el movimiento), se prohibe tres iteraciones y la variable x_2 solo estará prohibida dos iteraciones.
La nueva solución entra a la tablas de frecuencia. La nueva solución es: $S_0 = (0, 1, 1, 0, 0, 1, 1)$ *con* $z(S_0) = 74$

Cuadro 4.28: *Búsqueda tabú, iteración 3*

1	2	3	4	5	6	7	F. Obj.	Σa_i
0	1	1	0	0	1	1	74	39
0	1	1	0	1	1	1	81*	43
0	0	1	0	0	1	1	54	29
1	1	1	0	0	1	1	44	70
0	1	1	1	0	1	1	7	58
0	1	1	0	1	1	1	81	43

Lista tabú

1	2	3	4	5	6	7
0	1	0	0	3	2	0

Frecuencia

Solución	Frecuencia
$0, 1, 1, 0, 0, 0, 1$	1
$0, 1, 1, 0, 0, 1, 1$	1
$0, 1, 1, 0, 1, 1, 1$	1

Se actualiza la lista tabú: x_5 está prohibida tres iteraciones ya que es el movimiento reciente, y las demás disminuyen en una unidad la prohibición.
La nueva solución es:

$$S_0 = (0, 1, 1, 0, 1, 1, 1) \text{ con } z(S_0) = 81$$

En esta iteración se encontraron dos soluciones no factibles en los vecinos tres y cuatro, por esta razón se penalizaron al evaluar en la función objetivo, esto es:

$$z(X_3) = 144 - 100 = 44$$

$$z(X_34) = 107 - 100 = 7$$

Cuadro 4.29: *Búsqueda tabú, iteración 4*

1	2	3	4	5	6	7	F. Obj.	Σa_i
0	1	1	0	1	1	1	81	43
0	1	1	0	0	1	1	74	39**T**
0	1	1	0	1	1	0	71*	37
0	1	0	0	1	1	1	42	23
0	1	1	0	1	0	1	76	40**T**
0	1	1	0	1	1	0	71	37

Lista tabú

1	2	3	4	5	6	7
0	0	0	0	2	1	3

Frecuencia

Solución	Frecuencia
0, 1, 1, 0, 0, 0, 1	1
0, 1, 1, 0, 0, 1, 1	1
0, 1, 1, 0, 1, 1, 1	1
0, 1, 1, 0, 1, 1, 0	1

En esta iteración la mejor solución es el vecino cuatro pero el movimiento de la entrada seis está prohibido dos iteraciones más, pasamos a la siguiente mejor solución pero también es tabú (el movimiento de la variable x_5), pasamos a la siguiente mejor solución, esta es el segundo vecino, modificamos la lista tabú y la tabla de frecuencia. (Las soluciones tabú se marcarán con una T)

Cuadro 4.30: *Búsqueda tabú, iteración 5*

1	2	3	4	5	6	7	F. Obj.	Σa_i
0	1	1	0	1	1	0	71	37
0	1	0	0	1	1	0	32*	17
0	0	1	0	1	1	0	51	27**T**
0	1	1	0	1	1	1	81	43**T**
0	1	1	0	1	1	1	81	43**T**
0	1	1	1	1	1	0	8	56

Lista tabú

1	2	3	4	5	6	7
0	0	3	0	1	0	2

Frecuencia

Solución	Frecuencia
0, 1, 1, 0, 0, 0, 1	1
0, 1, 1, 0, 0, 1, 1	1
0, 1, 1, 0, 1, 1, 1	1
0, 1, 1, 0, 1, 1, 0	1
0, 1, 0, 0, 1, 1, 0	1

Cuadro 4.31: *Búsqueda tabú, iteración 6*

1	2	3	4	5	6	7	F. Obj.	Σa_i
0	1	0	0	1	1	0	32	17
1	1	0	0	1	1	0	102*	48
0	1	0	0	1	0	0	27	14
0	1	0	0	0	1	0	25	13**T**
0	1	1	0	1	1	0	71	37**T**
0	1	1	0	1	1	0	71	37**T**

Lista tabú

1	2	3	4	5	6	7
3	0	2	0	0	0	1

Frecuencia

Solución	Frecuencia
0, 1, 1, 0, 0, 0, 1	1
0, 1, 1, 0, 0, 1, 1	1
0, 1, 1, 0, 1, 1, 1	1
0, 1, 1, 0, 1, 1, 0	1
0, 1, 0, 0, 1, 1, 0	1
1, 1, 0, 0, 1, 1, 0	1

Cuadro 4.32: *Búsqueda tabú, iteración 7*

1	2	3	4	5	6	7	F. Obj.	Σa_i
1	1	0	0	1	1	0	102	48
1	1	1	0	1	1	0	41	68**T**
1	1	0	0	0	1	0	95	44
1	1	0	0	1	0	0	97*	45
1	1	0	0	1	0	0	97	45
1	1	0	0	0	1	0	95	44

Lista tabú

1	2	3	4	5	6	7
2	0	1	0	0	3	0

Frecuencia

Solución	Frecuencia
0, 1, 1, 0, 0, 0, 1	1
0, 1, 1, 0, 0, 1, 1	1
0, 1, 1, 0, 1, 1, 1	1
0, 1, 1, 0, 1, 1, 0	1
0, 1, 0, 0, 1, 1, 0	1
1, 1, 0, 0, 1, 1, 0	1
1, 1, 0, 0, 1, 0, 0	1

Cuadro 4.33: *Búsqueda tabú, iteración 8*

1	2	3	4	5	6	7	F. Obj.	Σa_i
1	1	0	0	1	0	0	97	45
1	1	0	1	1	0	0	34*	64
1	1	0	0	1	1	0	102	48**T**
1	1	0	0	1	0	1	7	51
1	1	1	0	1	0	0	36	65**T**
1	1	0	1	1	0	0	34	64

Lista tabú

1	2	3	4	5	6	7
1	0	0	3	0	2	0

Frecuencia

Solución	Frecuencia
0, 1, 1, 0, 0, 0, 1	1
0, 1, 1, 0, 0, 1, 1	1
0, 1, 1, 0, 1, 1, 1	1
0, 1, 1, 0, 1, 1, 0	1
0, 1, 0, 0, 1, 1, 0	1
1, 1, 0, 0, 1, 1, 0	1
1, 1, 0, 0, 1, 0, 0	1
1, 1, 0, 1, 1, 0, 0	1

Cuadro 4.34: *Búsqueda tabú, iteración 9*

1	2	3	4	5	6	7	F. Obj.	Σa_i
1	1	0	1	1	0	0	34	64
1	1	0	1	1	1	0	39	67**T**
1	0	0	1	1	0	0	14	54
0	1	0	1	1	0	0	64	33**T**
1	1	0	1	0	0	0	27*	60
1	1	0	1	0	0	0	27	60

Lista tabú

1	2	3	4	5	6	7
0	0	0	2	3	1	0

Frecuencia

Solución	Frecuencia
0, 1, 1, 0, 0, 0, 1	1
0, 1, 1, 0, 0, 1, 1	1
0, 1, 1, 0, 1, 1, 1	1
0, 1, 1, 0, 1, 1, 0	1
0, 1, 0, 0, 1, 1, 0	1
1, 1, 0, 0, 1, 1, 0	1
1, 1, 0, 0, 1, 0, 0	1
1, 1, 0, 1, 1, 0, 0	1
1, 1, 0, 1, 0, 0, 0	1

Cuadro 4.35: *Búsqueda tabú, iteración 10*

1	2	3	4	5	6	7	F. Obj.	Σa_i
1	1	0	1	0	0	0	27	60
0	1	0	1	0	0	0	57	29
1	0	0	1	0	0	0	107*	50
1	1	1	1	0	0	0	66	80
1	1	0	1	1	0	0	34	64**T**
0	1	0	1	0	0	0	57	29

Lista tabú

1	2	3	4	5	6	7
0	3	0	1	2	0	0

Frecuencia

Solución	Frecuencia
0, 1, 1, 0, 0, 0, 1	1
0, 1, 1, 0, 0, 1, 1	1
0, 1, 1, 0, 1, 1, 1	1
0, 1, 1, 0, 1, 1, 0	1
0, 1, 0, 0, 1, 1, 0	1
1, 1, 0, 0, 1, 1, 0	1
1, 1, 0, 0, 1, 0, 0	1
1, 0, 0, 1, 0, 0, 0	1
1, 1, 0, 1, 0, 0, 0	1
1, 1, 0, 1, 0, 0, 0	1

Como el criterio de parada se cumple, podemos concluir que la solución es:

$$X^* = (1, 0, 0, 1, 0, 0, 0,) \text{ con } z((X^*) = 107$$

Que coincidió con la solución óptima.
Con la aplicación del algoritmo, podemos hacer las siguientes observaciones:

- *Es importante penalizar las soluciones no factibles de tal manera que sean menos atractivas, pero no demasiado ya que podemos dejar fuera del análisis algunas regiones que tal vez tengan al óptimo, como pasó en nuestra implementación, en las iteraciones ocho y nueve las soluciones S_0 no eran factibles, pero tuvimos que pasar por ellas para llegar al óptimo en la iteración* 10.
- *El criterio de aspiración se puede aplicar al pasar a una solución que es tabú pero, se tiene que tener cuidado de no ser tan flexibles ya que podemos caer en un ciclo.*

- *En nuestra implementación no se utilizó la información (por el número de iteraciones) de la tabla de frecuencia pero si se realizarán más iteraciones se empezarían a repetir algunas soluciones, en este caso se debe decidir cual será la penalización (en el caso de que se realice) para no caer en los óptimos locales (hacerlos menos atractivos).*

Algunas modificaciones que se pueden aplicar al algoritmo de búsqueda tabú son:

- *Manera de buscar soluciones vecinas.*
- *Número de iteraciones que el movimiento realizado en la iteración permanecerán Tabú.*
- *Penalización sobre las soluciones no factibles.*
- *Agregar sub rutinas para convertir soluciones no factibles a soluciones factibles.*
- *El criterio de aspiración.*

4.8. Algoritmo genético

[De los Cobos, S. G. (2010) y Rodríguez, K.,(2018)] Los algoritmos genéticos fueron desarrollados por John Holland (1975), quien quería construir un esquema teórico basado en los principios de la evolución mediante la selección natural.
Los algoritmos genéticos se basan principalmente en dos características importantes de la evolución natural:

- La herencia genética que cada especie tiene, ésta debe mejorar de una generación a otra, ya que se deben desarrollar características que permitan a la siguiente generación una mejor adaptación a su medio ambiente para que pueda sobrevivir en él.
- La lucha por la supervivencia implica que cada especie se adapta de la mejor manera a su medio ambiente, el cual va cambiando con el tiempo haciéndose cada vez más complejo.

La información genética que se hereda de una generación a otra es a través de los genes, estos se transforman por el medio ambiente (el entorno) generando fenotipos, estos hacen que las nuevas generaciones tengan individuos mejores adaptados al entorno.

Figura 4.6: *Genotipo y fenotipo*

Fuente: Elaboración propia utilizando Geogebra 5 (2016)

Estos principios se puede adaptar a un problema de optimización, la información genética en un problema de optimización se almacenará en los genotipos (arreglos binarios), éstos contienen soluciones del problema codificadas, por esta razón es necesario decodificarlas (influencia del medio ambiente) en un fenotipo el cual ya tiene la solución del problema como un elemento del dominio de éste.

4.8.1. Tipos de genotipos

Los genotipos pueden tener diferentes codificaciones tales como:

- Binaria: La codificación binaria en problemas de optimización continua consiste en representar elementos del dominio de la función en el sistema de numeración binario y después ésta se traduce en un fenotipo que represente el valor numérico del dominio en la función, en problemas combinatorios existen 2 casos, si el problema es entero binario, entonces la codificación del genotipo coincide con la del fenotipo; si el problema es entero, se le pueden asignar valores a los números binarios tantos como sean necesarios y después transformarlo al fenotipo.

- Entera: La codificación entera, se utiliza en los problemas discretos, es decir, en los combinatorios, la decodificación es la del sistema de numeración binario.

- Decimal(Real): Esta codificación se puede utilizar en ambos problemas de optimización (continuos y discretos), es importante definir adecuadamente la decodificación para cada problema. En este tipo de codificación para los problemas con variables continuas se elimina la rutina de decodificación

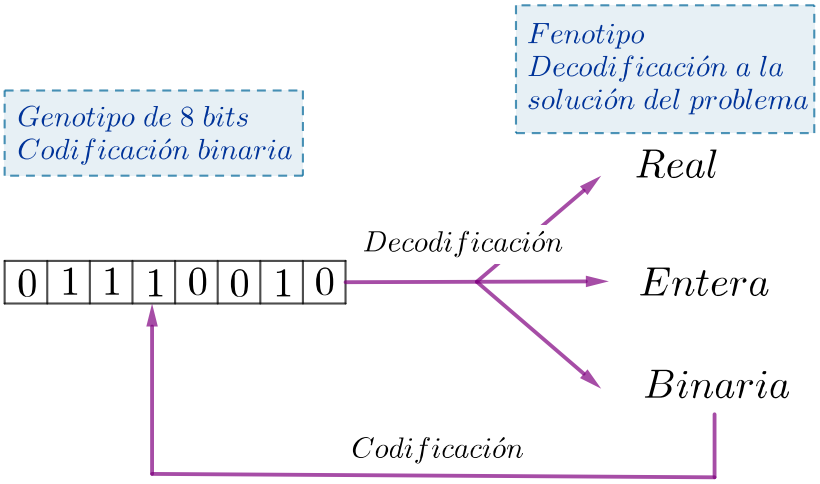


Figura 4.7: *Genotipo y fenotipo en un problema de optimización*

Fuente: Elaboración propia utilizando Geogebra 5 (2016)

4.8.2. Procedimiento general

Desde el inicio se debe fijar el número de individuos que tendrá cada generación, así como el criterio de parada.
Empezamos a generar individuos de manera aleatoria hasta llenar a la primer generación, estos se evaluarán en la función objetivo para obtener su aptitud, a partir de esto se puede aplicar elitismo, esto es, pasar a cierto porcentaje de los individuos con mejor aptitud a la siguiente generación (esto nos garantiza que los más aptos estarán en la siguiente generación), después se aplica el método de selección (por torneo) para obtener la pareja de posibles padres a los que se les aplicarán los operadores genéticos para completar a los individuos de la siguiente generación, mientras el criterio de parada no se cumpla este procedimiento se repite, cuando el criterio de parada se cumpla, se escoge al individuo con mejor aptitud de la última generación, éste será el óptimo.

4.8.3. Operadores genéticos

[Marcos (2010) y Rodríguez, K.,(2018)]. Los operadores genéticos tienen la función de reproducir la población seleccionada mediante el cruzamiento y la mutación.

- La cruza es un proceso de combinación combinación de fragmentos de cromosomas entre sí.

- La mutación introduce y mantiene la diversidad genética en la población con baja probabilidad de obtener una nueva solución de alta probabilidad.

Los operadores genéticos se aplican siempre y cuando cumplan con cierta probabilidad P_c (Probabilidad de que los padres se crucen) y P_m(Probabilidad de que los hijos, resultado del cruzamiento, se muten), usualmente la probabilidad de cruza es grande y la de mutación es pequeña.

Operadores genéticos para el problema de la mochila

Los operadores genéticos son distintos dependiendo del problema que se está resolviendo, para el caso del problema de la mochila binario **la cruza por un punto** consiste en seleccionar a dos individuos de la población que se denominan padres, ambos heredan genes a sus hijos, éste es un procedimiento de recombinación, para realizarlo, se genera un número aleatorio r en el intervalo $[0, \textit{longitud del individuo}]$ que será el punto del cruzamiento; a partir de la entrada r se intercambian las colas de los cromosomas (individuos) para formar dos individuos nuevos.

Ejemplo 19 *Supongamos que $r = 4$, entonces, las colas de los arreglos se intercambian a partir de la entrada* 4*, esto es:*

Figura 4.8: *Operador genético: Cruza*

La mutación consiste en generar un número aleatorio q en el intervalo $[0, \textit{longitud del individuo}]$ que será el punto de mutación, a partir de la entrada q se intercambian las entradas con ceros por unos y los unos por ceros, cabe señalar que la mutación se aplicará sobre individuos que ya hayan pasado por el operador genético de cruza.

Ejemplo 20 *Supongamos que $q = 5$, entonces, a partir de la entrada* 5 *se intercambian los ceros por unos y viceversa. En la siguiente figura se ilustra la mutación.*

Figura 4.9: *Operador genético: Mutación*

En el siguiente diagrama de flujo se explican los pasos a seguir al aplicar el algoritmo genético al un problema de la mochila binario:

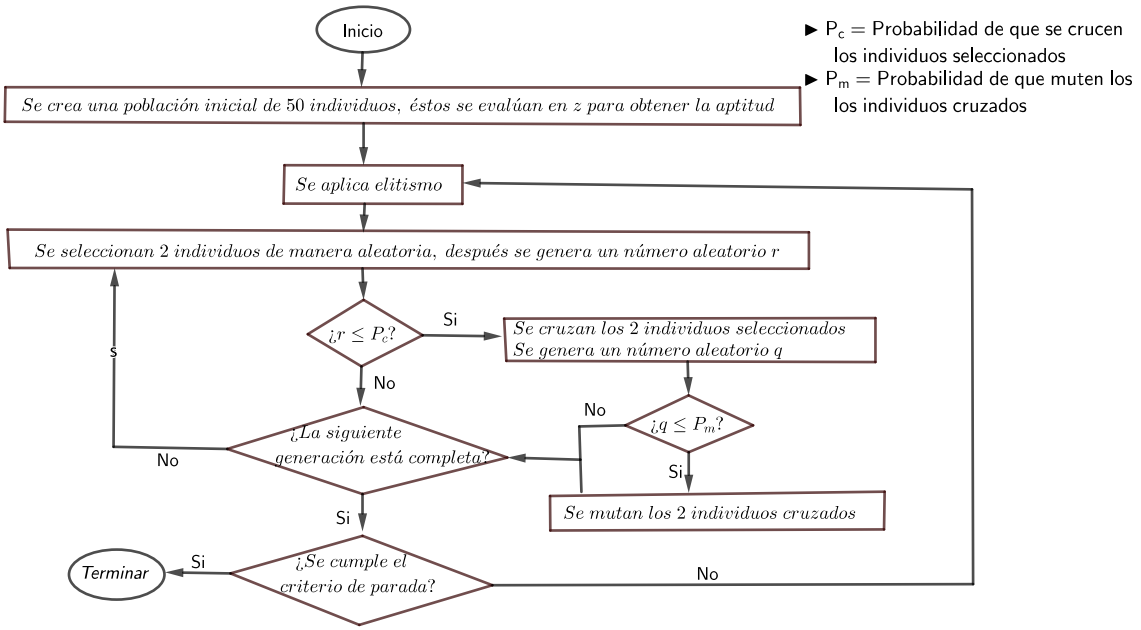


Figura 4.10: *Algoritmo Génetico para el problema binario*

Fuente: Elaboración propia utilizando Geogebra 5 (2016)

4.8.4. Algoritmo genético aplicado al problema de la mochila

Primero definiremos los elementos fundamentales del algoritmo:

- Cada generación tendrá 50 individuos.
- El método de selección será el de torneo determinista (siempre se escoge al mejor individuo).
- La probabilidad de cruza es: $P_c = 0.8$
- La probabilidad de mutación es: $P_m = 0.01$

Objetivo: Obtener una solución factibles y "óptima"para una instancia del problema de la mochila

1. Generar una población inicial de manera aleatoria.
2. Evaluar cada uno de los individuos (genes) en la función objetivo para obtener la aptitud de cada individuo.
3. Aplicar elitismo (es opcional) Se recomienda que pase a la siguiente generación un pequeño porcentaje de los individuos con mejor aptitud, alrededor del 10 % , esto es para no estancarse en los óptimos locales.

4. Aplicar el método de selección (nosotros aplicaremos por torneo determinista, esto es que de 2 individuos se compara su aptitud y se escoge al mejor).

5. A los individuos seleccionados se les aplican los operadores genéticos con la probabilidad descrita anteriormente, el objetivo de estos operadores es completar el número de individuos de la siguiente generación.

6. Si el criterio de parada se cumple, terminar, la solución óptima es la mejor solución de la última generación; en caso contrario ir a 2.

Ejemplo 21 *Se aplicará este algoritmo a la instancia con* $n = 8$*, los datos son:*

$$c = [350, 400, 450, 20, 70, 8, 5, 5]$$

$$a = [25, 35, 45, 5, 25, 3, 2, 2]$$

$$b = 104$$

El modelo matemático correspondiente a esta instancia es:

$$\begin{gathered} maz\ z = 350x_1 + 400x_2 + 450x_3 + 20x_4 + 70x_5 + 8x_6 + 5x_7 + 5x_8 \\ sujeto\,a \\ 25x_1 + 35x_2 + 45x_3 + 5x_4 + 25x_5 + 3x_6 + 2x_7 + 2x_8 \leq 104 \\ x_i \in \{0, 1\}\ \forall i \end{gathered}$$

*Generamos la población inicial: (*10 *individuos)*
Primer generación

$(0,0,1,0,1,0,1,0), (0,1,0,0,0,0,0,1), (1,1,0,0,0,0,0,1), (1,0,0,1,1,1,1,1), (1,0,1,1,1,1,0,0)$

$(1,0,0,0,1,1,1,1), (0,0,0,0,1,0,0,0), (1,1,1,1,1,0,0,1), (0,1,1,0,1,0,1,0), (0,0,1,1,1,1,0,1)$

Obtenemos la aptitud de los genes (individuos)

$$462, 405, 755, 395, 835, 375, 7, -3768, -4138, 490$$

Empezamos a obtener la población de la 2*a generación.*
Aplicamos elitismo y un 10 % *de la población inicial pasa a la siguiente generación, el individuo con mejor aptitud, este es:* $X_5^1 = (1,0,1,1,1,1,0,0) = X_1^2$ *con* $z(X_1^2) = 835$
La notación que utilizaremos es: el subíndice es el número de individuo y el hiperíndice es la generación.
Los 9 *individuos restantes se completarán con el cruzamiento y la mutación. Para cada cruce y mutación se deben elegir* 2 *padres utilizando el método de selección por torneo*

- *Selección de los primeros padres: Generamos* r_1 *y* r_2 *para elegir al primer padre;* $r_1 = 4\ yr_2 = 3$ *las aptitudes a comparar son:*

$$z(X_4^1)\ y\ z(X_3^1)$$

$$395 < 755$$

El primer padre es $X_3^1 = (1, 1, 0, 0, 0, 0, 0, 1)$
Para elegir al segundo padre generamos $r_1 = 5$ *y* $r_2 = 6$*, comparamos sus aptitudes*

$$z(X_5^1) \text{ y } z(X_6^1)$$

$$835 > 375$$

El segundo padre es: $X_5^1 = (1, 0, 1, 1, 1, 1, 0, 0)$
Generamos $r_3 \in [0, 1]$*, si* $r_3 \leq 0.8$ *(probabilidad de cruza) se cruzan los padres.*

$$r_3 = 0.5960 < 0.8$$

Obtenemos r_4 *para saber desde qué entrada se intercambian las entradas,* $r_4 = 2$
Los hijos son:

$$X_2^2 = (1, 0, 1, 1, 1, 1, 0, 0) \text{ y } X_3^2 = (1, 1, 0, 0, 0, 0, 0, 1)$$

Obtenemos su aptitud:

$$z(X_2^2) = 835 \text{ y } z(X_3^2) = 755$$

Se genera otro número aleatorio para ver si se mutan, sea $r_5 = 0.19 > 0.01$ *por lo tanto no se mutan.*

- *Selección de la segunda pareja de padres: Generamos* r_1 *y* r_2 *para elegir al primer padre;* $r_1 = 3$ *y*$r_2 = 1$ *las aptitudes a comparar son:*

$$z(X_3^1) \text{ y } z(X_1^1)$$

$$755 > 462$$

El primer padre es $X_3^1 = (1, 1, 0, 0, 0, 0, 0, 1)$
Para elegir al segundo padre generamos $r_1 = 9$ *y* $r_2 = 1$*, comparamos sus aptitudes*

$$z(X_9^1) \text{ y } z(X_1^1)$$

$$-4138 < 462$$

El segundo padre es: $X_1^1 = (0, 0, 1, 0, 1, 0, 1, 0)$
Generamos $r_3 = 0.29 < 0.8$*, se cruzan.* $r_4 = 8$*, entonces los hijos son:*

$$X_4^2 = (1, 1, 0, 0, 0, 0, 0, 0) \text{ y } X_5^2 = (0, 0, 1, 0, 1, 0, 1, 1)$$

$$z(X_4^2) = 750 \text{ y } z(X_5^2) = 467$$

Se genera otro número aleatorio para ver si se mutan, sea $r_5 = 0.457 > 0.01$ *por lo tanto no se mutan.*

- *Selección de la tercer pareja de padres: Generamos* r_1 *y* r_2 *para elegir al primer padre;* $r_1 = 6$ *y*$r_2 = 7$ *las aptitudes a comparar son:*

$$z(X_6^1) \text{ y } z(X_7^1)$$

$$375 > 7$$

El primer padre es $X_6^1 = (1, 0, 0, 0, 1, 1, 1, 1)$
Para elegir al segundo padre generamos $r_1 = 9$ *y* $r_2 = 8$*, comparamos sus aptitudes*

$$z(X_9^1) \text{ y } z(X_8^1)$$

$$-4138 < -3768$$

El segundo padre es: $X_8^1 = (1, 1, 1, 1, 1, 0, 0, 1)$
Generamos $r_3 = 0.45 < 0.8$*, se cruzan.* $r_4 = 7$*, entonces los hijos son:*

$$X_6^2 = (1, 0, 0, 0, 1, 1, 0, 1) \text{ y } X_7^2 = (1, 1, 1, 1, 1, 0, 1, 1)$$

$$z(X_6^2) = 370 \text{ y } z(X_7^2) = -3763$$

Se genera otro número aleatorio para ver si se mutan, sea $r_5 = 0.98 > 0.01$ *por lo tanto no se mutan.*

- *Selección de la cuarta pareja de padres, generamos* $r_1 = 8$ *y*$r_2 = 10$ *las aptitudes a comparar son:*

$$z(X_8^1) \text{ y } z(X_{10}^1)$$

$$-3768 < 490$$

El primer padre es $X_{10}^1 = (0, 0, 1, 1, 1, 1, 0, 1)$
Para elegir al segundo padre generamos $r_1 = 6$ *y* $r_2 = 1$*, comparamos sus aptitudes*

$$z(X_1^1) \text{ y } z(X_9^1)$$

$$375 < 462$$

El segundo padre es: $X_9^1 = (0, 0, 1, 0, 1, 0, 1, 0)$
Generamos $r_3 = 0.66 < 0.8$*, se cruzan.* $r_4 = 5$*, entonces los hijos son:*

$$X_8^2 = (0, 0, 1, 1, 1, 0, 1, 0) \text{ y } X_9^2 = (0, 0, 1, 0, 1, 1, 0, 1)$$

$$z(X_9^2) = 482 \text{ y } z(X_{10}^2) = 470$$

Se genera otro número aleatorio para ver si se mutan, sea $r_5 = 0.37 > 0.01$ *por lo tanto no se mutan.*

- *Para la quinta pareja de padres generamos* $r_1 = 5$ *y*$r_2 = 3$ *las aptitudes a comparar son:*

$$z(X_5^1) \text{ y } z(X_3^1)$$

$$835 < 755$$

El primer padre es $X_3^1 = (1, 0, 1, 1, 1, 1, 0, 0)$
Para elegir al segundo padre generamos $r_1 = 4$ *y* $r_2 = 2$*, comparamos sus aptitudes*

$$z(X_1^1) \; y \; z(X_9^1)$$

$$395 < 405$$

El segundo padre es: $X_9^1 = (0, 1, 0, 0, 0, 0, 0, 1)$
Generamos $r_3 = 0.44 < 0.8$*, se cruzan.* $r_4 = 4$*, entonces los hijos son:*

$$X_{10}^2 = (1, 0, 1, 0, 0, 0, 0, 1) \; y \; X_{11}^2 = (0, 1, 0, 1, 1, 1, 0, 0)$$

$$z(X_{10}^2) = 805 \; y \; z(X_{11}^2) = 435$$

Se genera otro número aleatorio para ver si se mutan, sea $r_5 = 0.637 > 0.01$ *por lo tanto no se mutan.*
Como en las iteraciones se generaron 11 *genes (individuos) y cada generación solo contiene* 10*, omitiremos al individuo com menos aptitud, esta es* X_7^2 *por los que la segunda generación es:*

Segunda generación

$(1, 0, 1, 1, 1, 1, 0, 0), (1, 0, 1, 1, 1, 1, 0, 0), (1, 1, 0, 0, 0, 0, 0, 1), (1, 1, 0, 0, 0, 0, 0, 0), (0, 0, 1, 0, 1, 0, 1, 1)$

$(1, 0, 0, 0, 1, 1, 0, 1), (0, 0, 1, 1, 1, 0, 1, 0), (0, 0, 1, 0, 1, 1, 0, 1), (1, 0, 1, 0, 0, 0, 0, 1), (0, 1, 0, 1, 1, 1, 0, 0)$

Obtenemos la aptitud de los genes (individuos)

$$835, 835, 755, 750, 467, 370, 482, 470, 805, 435$$

Las iteraciones para generar a la tercer generación se resumen en el cuadro 4.1*: Como en las iteraciones se generaron* 11 *genes (individuos) y cada generación solo contiene* 10*, omitiremos al individuo com menos aptitud, esta es* X_8^3 *por los que la tercer generación es:*
Tercer generación

$(1, 0, 1, 1, 1, 1, 0, 0), (1, 0, 1, 1, 1, 1, 0, 1), (1, 0, 1, 0, 0, 1, 0, 0), (1, 0, 1, 1, 1, 1, 0, 0), (1, 0, 1, 1, 1, 1, 0, 0)$

$(1, 1, 0, 0, 1, 1, 0, 0), (1, 0, 1, 1, 0, 0, 0, 1), (1, 1, 0, 0, 0, 0, 0, 0), (0, 0, 1, 1, 1, 1, 0, 0), (1, 0, 1, 0, 1, 1, 0, 1)$

Obtenemos la aptitud de los genes (individuos)

$$835, 832, 808, 835, 835, 765, 825, 750, 485, 820$$

Las iteraciones para generar a la cuarta generación se resumen en el siguiente cuadro 4.2*:*

Utilizando la información de la tabla 4.2 *podemos desarrollar lo siguiente: como en la última iteración el criterio para realizar la mutación se cumple como igualdad realizaremos la mutación de las últimas soluciones, primero generamos un número aleatorio* $r = 5$*, éste nos indicara a partir de qué entrada se intercambiarán los* $1's$

Cuadro 4.36: *Iteraciones para obtener la tercer generación*

Padre 1 r_1 y r_2	$z(X_{r1})\_z(X_{r2})$ Padre 1	Padre 2 r_1 y r_2	$z(X_{r1})\_z(X_{r2})$ Padre 2	¿$r_3 < P_c$?	Entrada parar cruzar	Hijos Aptitud	¿$r_4 < P_m$?
1 y 8	$835 > 470$ $(1,0,1,1,1,1,0,0)$	9 y 8	$805 > 470$ $(1,0,1,0,0,0,0,1)$	$.55 < .8$	6	$(1,0,1,1,1,1,0,0,1)$ 832 $(1,0,1,0,0,1,0,0)$ 808	$0.15 > 0.01$
2 y 5	$835 > 467$ $(1,0,1,1,1,1,0,0)$	1 y 9	$835 > 805$ $(1,0,1,1,1,1,0,0)$	$.47 < .8$	5	$(1,0,1,1,1,1,0,0)$ 835 $(1,0,1,1,1,1,0,0)$ 835	$0.77 > 0.01$
10 y 5	$435 < 467$ $(0,0,1,0,1,0,1,1)$	6 y 9	$370 < 805$ $((1,0,1,0,0,0,0,1)$	$.85 < .8$	___	_	_
3 y 10	$755 > 435$ $(1,1,0,0,0,0,0,1)$	2 y 5	$835 > 467$ $(1,0,1,1,1,1,0,0)$	$.33 < .8$	5	$(1,1,0,0,1,1,0,0)$ 765 $(1,0,1,1,0,0,0,1)$ 825	$0.33 > 0.01$
2 y 7	$835 > 482$ $(1,0,1,1,1,1,0,0)$	3 y 4	$755 > 750$ $(1,1,0,0,0,0,0,1)$	$.23 < .8$	7	$(1,0,1,1,1,1,0,1)$ -4160 $(1,1,0,0,0,0,0,0)$ 750	$0.18 > 0.01$
8 y 5	$470 > 467$ $(0,0,1,0,1,1,0,1)$	4 y 1	$750 < 835$ $(1,0,1,1,1,1,0,0)$	$.22 < .8$	2	$(0,0,1,1,1,1,0,0)$ 485 $(1,0,1,0,1,1,0,1)$ 820	$0.502 > 0.01$

Cuadro 4.37: *Iteraciones para obtener la cuarta generación*

Padre 1 r_1 y r_2	$z(X_{r1})\_z(X_{r2})$ Padre 1	Padre 2 r_1 y r_2	$z(X_{r1})\_z(X_{r2})$ Padre 2	¿$r_3 < P_c$?	Entrada parar cruzar	Hijos Aptitud	¿$r_4 < P_m$?
7 *y* 4	$825 < 835$ $(1,0,1,1,1,1,0,0)$	8 *y* 3	$750 < 808$ $(1,0,1,0,0,1,0,0)$	$.2 < .8$	5	$(1,0,1,1,0,1,0,0)$ 828 $(1,0,1,0,1,1,0,0)$ 815	$0.09 > 0.01$
3 *y* 1	$808 < 835$ $(1,0,1,1,1,1,0,0)$	2 *y* 4	$832 < 835$ $((1,0,1,1,1,1,0,0)$	$.88 < .8$	——	—	—
10 *y* 5	$820 < 835$ $(1,0,1,1,1,1,0,0)$	1 *y* 9	$835 > 485$ $(1,0,1,1,1,1,0,0)$	$.66 < .8$	3	$(1,0,1,1,1,1,0,0)$ 835 $(1,0,1,1,1,1,0,0)$ 835	$0.92 > 0.01$
2 *y* 10	$832 > 820$ $(1,0,1,1,1,0,0,1)$	8 *y* 2	$750 < 832$ $(1,0,1,1,1,0,0,1)$	$.49 < .8$	2	$(1,0,1,1,1,1,0,0,1)$ 832 $(1,0,1,1,1,1,0,0,1)$ 832	$0.43 > 0.01$
4 *y* 6	$835 > 765$ $(1,0,1,1,1,1,0,0)$	9 *y* 2	$485 < 832$ $(1,0,1,1,1,1,0,0,1)$	$.69 < .8$	3	$(1,0,1,1,1,1,0,0,1)$ 832 $(1,0,1,1,1,1,1,0,0)$ 835	$0.01 = 0.01$

por $0's$ *y viceversa; dando como resultado:*

$$X_{10}^4 = (1,0,1,1,0,1,1,0);\ z(X_{10}^4) = 835$$
$$X_{11}^4 = (1,0,1,1,0,0,1,1);\ z(X_{11}^4) = 830$$

Omitimos al individuo com menos aptitud, este es X_{10}^4 *por los que la cuarta generación es:*

Cuarta generación

$(1,0,1,1,1,1,0,0), ((1,0,1,1,1,1,0,0), (1,0,1,0,0,1,0,0), (1,0,1,1,1,1,0,0), (1,0,1,1,1,1,0,0)$

$(1,0,1,1,1,0,0,1), (1,0,1,1,1,0,0,1), (1,0,1,1,1,0,0,1), (1,0,1,1,1,1,0,0), (1,0,1,1,0,0,1,1)$

Obtenemos la aptitud de los genes (individuos)

$$835, 828, 815, 835, 835, 832, 832, 832, 835, 835$$

Podemos observar que a pesar de que el número de individuos es pequeño (10) *la aptitud se está encasillando, es decir, todas las soluciones tienen aptitud mayor a* 800.

Podemos concluir que en el algoritmo genético es de suma importancia el número de individuos por generación para salir de los óptimos locales ya que en nuestro ejemplo nos estamos estancando en un óptimo local, la diferencia la puede hace el operador genético mutación iya que puede ser una salida de los óptimos locales, aunque es importante que la probabilidad de mutación sea pequeña para tener abierto el camino de explorar otras regiones sin dejar a un lado que los individuos más aptos son los que deben prevalecer para alcanzar objetivo que es maximizar z.

Implementación

Aplicamos el algoritmo genético a las 5 instancias anteriores con los siguientes elementos:

- Cada generación tendrá 50 individuos
- La probabilidad de cruza es 0.8
- La probabilidad de mutación es de 0.01
- El número de iteraciones en que se aplicará es para : 50, 100 y 500

Cuadro 4.38: *Algoritmo genético, instancia 1*

N^o de generaciones	N^ode veces que cae en el óptimo después de 50 corridas
50	45
100	43
500	44

Cuadro 4.39: *Algoritmo genético, instancia 2*

N^o de generaciones	N^ode veces que cae en el óptimo después de 50 corridas
50	33
100	41
500	37

Cuadro 4.40: *Algoritmo genético, instancia 3*

N^o de generaciones	N^ode veces que cae en el óptimo después de 50 corridas
50	13
100	18
500	15

Cuadro 4.41: *Algoritmo genético, instancia 4*

N^o de generaciones	N^ode veces que cae en el óptimo después de 50 corridas
50	34
100	34
500	41

Cuadro 4.42: *Algoritmo genético, instancia 5*

N^o de generaciones	Mejor valor de z después de 100 corridas
50	$13,404,042$
100	$13,392,791$
500	$13,406,857$

La mejor solución encontrada con este algoritmo es: $13,406,857$, tomando en cuenta este valor, la efectividad del algoritmo es:

$$\%\, efectividad = 100 - \left[\left(\frac{13,549,094 - 13,406,857}{13,549,094}\right) \times 100\right] = 98.9502\,\%$$

Podemos observar que el algoritmo genético en comparación de los 3 algoritmos que se implementaron para las instancias del problema de la mochila es el mejor, ya que cae un mayor número de veces en el óptimo después de 50 corridas, además el

porcentaje de efectividad para la instancia 5 fue el mejor.
En la implementación de este algoritmo en las instancias anteriores podemos observar que el número de generaciones no es un factor determinante para llegar al óptimo, ya que las soluciones vecinas de un problema combinatorio no se comporta de manera estable, una permutación en una solución puede llevarnos a una solución bastante mala.

Como todos los algoritmos metaheurísticos, es posible modificarlo para mejorar el resultado obtenido, algunos de estos cambios son propuestos en los siguientes artículos:

- En el artículo de Yan, T. S., Guo, G. Q., Li, H. M. proponen manejar dos tipos de poblaciones, las llaman: Población seleccionada que es la población principal y la población eliminada por selección y la denominan como población subordinada.
  Los operadores genéticos los manejan de manera dual, esto es:
  - Cruce dual: Realizan un cruce de la población principal y otro de la población subordinada. (La probabilidad para realizar el cruce de la población principal es alta, mientras que para la población subordinada, la probabilidad de cruce es baja)
  - Mutación dual: Realizan una mutación de la población principal y otra de la población subordinada. (La probabilidad para realizar la mutación de la población principal es baja, mientras que para la población subordinada, la probabilidad de mutación es alta)

  Las probabilidades las obtienen mediante la aptitud máxima y mínima de la población principal y la población subordinada.
- En el artículo de Yadav, R. K., Gupta, H., Jhingran, H., Agarwal, A. proponen un métodos diferente para seleccionar, este es el método de la ruleta y concluyen que utilizando este algoritmo es más eficiente que ramificación y acotamiento y programación dinámica.
- En el artículo de Hristakeva, M., & Shrestha, D. proponen convertir las soluciones no factibles a soluciones factibles, esto lo realizan cambiando un 1 por un 0 de manera aleatoria hasta que se cumpla la restricción de capacidad, también proponen dos métodos de selección (Ruleta y lo que denominan selección de grupos). En las conclusiones mencionan que el elitismo es un factor que diferencía ambos métodos de selección: Si no se utiliza elitismo, entonces es mejor utilizar el método de selección de grupo, pero si se aplica elitismo, entonces es indiferente el método de selección.

Conclusiones

Al termino del trabajo de investigación podemos concluir que a pesar de que el modelo matemático del problema de la mochila es sencillo, el problema de decisión es NP-completo, es decir, el problema de optimización es NP-duro, a pesar de esto es un problema muy estudiado y por esta razón dedicamos un capítulo a la taxonomía del problema de la mochila que si bien, no se enlistaron todas las variantes, las que se describieron se llevaron a la par con un problema contextualizado para poder identificar cómo influyen las nuevas restricciones en el modelo matemático

El tema de la NP-completez se manejó de manera general; aunque la reducción polinomial llega hasta el problema de la mochila puede tomarse como base para otras reducciones y así aplicar lo desarrollado en este trabajo en otros problemas de optimización. Al manejar algoritmos y métodos de solución de manera general, este trabajo de investigación puede ser una referencia para los estudiantes que cursen la asignatura de programación entera a nivel licenciatura o posgrado ya que nos enfocamos a explicar detalladamente cada uno de los algoritmos utilizados: exactos, heurísticos y metaheurísticos; de los dos últimos se compararon resultados y se concluyó acerca de la efectividad de cada uno de ellos con la finalidad de que se tengan elementos suficientes para poder decidir qué algoritmo utilizar e incluso se citan algunos artículos que pueden servir de referencia para mejorar el resultado de los algoritmos propuestos en este trabajo.

Una de las bondades de los algoritmos metaheurísticos es que se pueden adaptar (modificar) de manera interactiva, esto genera que haya más estudios asociados a estos algoritmos y cada vez se mejoren las 3 $E's$ Efectivo, eficaz y eficiente, y como consecuencia se obtengan mejores soluciones a los problemas en menor tiempo. Es importante tomar en cuenta que para poder realizar dichas modificaciones se deben conocer las características y criterios que se aplican en cada uno de los algoritmos, de esta manera se prodrán proponer nuevos procedimientos (sub rutinas) que nos acerquen más al óptimo, por ejemplo: Al implementar el algoritmo genético a instancias de 50 artículos obtenida de :
`http://hjemmesider.diku.dk/~pisinger/codes.html` obtenemos resultados muy variados, tales como:

- Soluciones no factibles.
- Soluciones de mala calidad.

La penalización es una herramienta muy útil en instancias pequeñas o tal vez en instancias en las que la diferencia entre los coeficientes de z no sea muy grande, ya

que la penalización se puede asociar al valor más grande de los coeficientes de z; por ejemplo, si el coeficiente más pequeño es $c_1 = 8$ y $c_9 = 5000$ es el coeficiente más grande, entonces, la penalización puede ser 5000 para las soluciones no factibles, al aplicarla podemos caer en dos casos :

- Si $x_9 = 1$ entonces la penalización podría cumplir con su objetivo (hacer menos atractiva a esta solución)
- Si $x_9 = 0$, entonces, esto nos puede llevar a una solución con $z < 0$ o una aptitud muy baja que haría menos atractivo a este individuo, en ocasiones sería difícil salir de este tipo de soluciones.

Por esta razón es importante conocer el problema que deseamos resolver ya que en ocasiones es posible manejar las soluciones no factibles con penalizaciones, pero existen instancias para las que es necesario una sub rutina que modifique las soluciones no factibles para que cumplan con las restricciones del problemas y no caer estancado en este tipo de soluciones

Al utilizar la implementación de los algoritmos heurísticos y metaheurísticos propuestos en este trabajo a diferentes instancias (hasta de 24 artículos) del problema de la mochila obtuvimos la siguiente información sobre la efectividad de estos:

Algoritmo	Efectividad
Glotón (Heurístico)	95.83 %
Búsqueda en su entorno (Heurístico)	97.92 %
Recocido Simulado (Metaheurístico)	98.7767 %
Genético (Metaheurístico)	98.9502 %

Por lo que podemos concluir que el algoritmo genético propuesto en este trabajo es el que tiene mayor efectividad.

Si además se busca que el algoritmo sea eficiente podríamos inclinarnos por recocido simulado, ya que este algoritmo no utiliza tanta memoria para su implementación. Para un trabajo de investigación futuro se pueden hacer diferentes modificaciones a los algoritmos aquí propuestos para comparar los resultados e incluir un análisis sobre la eficiencia de éstos.

Referencias bibliográficas

- Ahuja, R. K., Magnanti, T. L., Orlin, J. B. (1988). Network flows.
- Baker, B. S., Coffman Jr, E. G. (1996). Mutual exclusion scheduling. Theoretical Computer Science, 162(2), 225-243.
- Bertsekas Dimitri P. 1987. Dynamic Programming, Deterministic and Stochastic Models. Ed. Prentice-Hall, INC, USA,
- Dantzig, G. B. (1957). Discrete-variable extremum problems. Operations research, 5(2), 266-288.
- De los Cobos, S. G. (2010). Búsqueda y exploración estocástica. Universidad Autónoma Metropolitana.
- de Antonio Suárez, O. (2011). Una aproximación a la heuristica y metaheuristicas. INGE@ UAN-Tendencias en la Ingeniería, 1(2).
- Dowsland, K. A., Díaz, B. A. (2003). Diseño de heurística y fundamentos del recocido simulado. Inteligencia Artificial. Revista Iberoamericana de Inteligencia Artificial, 7(19), 0.
- Glover, F. (1986). Future paths for integer programming and links to artificial intelligence. Computers operations research, 13(5), 533-549.
- Glover, F., & Melián, B. (2003). Búsqueda tabú. Inteligencia Artificial. Revista Iberoamericana de Inteligencia Artificial, 7(19), 0.
- Hernández A,. 1994 Programación Dinámica. Publicaciones del Departamento de Matemáticas de la Facultad de Ciencias UNAM. Vínculos Matemáticos 206, México.
- Hernández A. (2013) Introducción a la programación lineal. *Las prensas de ciencias* 1*a* reimpresión. México UNAM.
- Hristakeva, M., & Shrestha, D. (2004, April). Solving the 0-1 knapsack problem with genetic algorithms. In Midwest instruction and computing symposium (pp. 16-17).
- Marcos; Rivero Gestal (Daniel; Rabuñal, Juan Ramón; Dorado, Julián; Pazos, Alejandro), Gestal, M. (2010). Introducción a los algoritmos genéticos y la programación genética (p. 32). Coruña: Universidad da Coruña.

- Martí, R. (2003). Procedimientos metaheurísticos en optimización combinatoria. Matemátiques, Universidad de Valencia, 1(1), 3-62.
- Melián, B., Pérez, J. A. M., & Vega, J. M. M. (2003). Metaheurísticas: Una visión global. Inteligencia Artificial. Revista Iberoamericana de Inteligencia Artificial, 7(19), 0.
- Moradi, N., Kayvanfar, V., & Rafiee, M. (2021). An efficient population-based simulated annealing algorithm for 0-1 knapsack problem. Engineering with Computers, 1-20.
- Papadimitriou, C. H., & Steiglitz, K. (1998). Combinatorial optimization: algorithms and complexity. Courier Corporation.
- Prawda Witenberg, J. (1976). Métodos y modelos de investigación de operaciones. Vol. I. Editorial Limusa SA, México
- Rodríguez, K.,(2018), *Apuntes del curso Métodos Heurísticos*, CDMX, México.
- Vélez, M. C., & Montoya, J. A. (2007). Metaheurísticos: una alternativa para la solución de problemas combinatorios en administración de operaciones. Revista Eia, (8), 99-115.
- Yadav, R. K., Gupta, H., Jhingran, H., Agarwal, A., & Gupta, A. (2017). An enhanced genetic algorithm to solve 0/1 knapsack problem. International Journal of Computer Science and Information Security (IJCSIS), 15(5).
- Yan, T. S., Guo, G. Q., Li, H. M., & He, W. (2013). A genetic algorithm for solving knapsack problems based on adaptive evolution in dual population. In Advanced Materials Research (Vol. 756, pp. 2799-2802). Trans Tech Publications Ltd.
- Zemel, E. (1980). The linear multiple choice knapsack problem. Operations Research, 28(6), 1412-1423.

Bibliografía

[1] Cook, S. A. (1971) *The complexity of theorem-proving procedures.* In Proceedings of the third annual ACM symposium on Theory of computing (pp. 151-158).

[2] Garey M., Johnson D., (1979) *Computers and intractability. A guide to the theory of NP-Completeness.* W.H. Freeman and company New York.

[3] Gendreau, M., Potvin, J.,(2010) *Handbook of Metaheuristics.* Segunda Edición, Springer.

[4] Glover, F., & Taillard, E. (1993) *A user's guide to tabu search.* Annals of operations research, 41(1), 1-28.

[5] Hernández A,. (2006) *.Apuntes de Programación Entera.* CDMX, México.

[6] Ibaraki, T., Hasegawa, T., Teranaka, K., & Iwase, J. (1978). *The multiple-choice knapsack problem.* Journal of the Operations Research Society of Japan, 21(1), 59-95.

[7] Jiménez-Castellano, I., Hernández-Ocaña, B., Hernández-Torruco, J., & Chávez-Bosquez, O. (2019) *Metaheuristics-based frameworks to solve the knapsack problem.* Revista INGENIERÍA UC, 26(1), 31-43.

[8] Karp, R. M. (1972). *Reducibility among combinatorial problems.* In Complexity of computer computations (pp. 85-103). Springer, Boston, MA.

[9] Parra, J. F. (1974). *Investigación de operaciones.* Boletín de Matemáticas, 8(1-6), 170-197.

[10] Posner, M. E., & Guignard, M. (1978). *The collapsing 0-1 knapsack problem.* Mathematical Programming, 15(1), 155-161.

[11] Puchinger, J., Raidl, G. R., & Pferschy, U. (2010). *The multidimensional knapsack problem: Structure and algorithms.* INFORMS Journal on Computing, 22(2), 250-265.

[12] Salkin Harvey M. 1975 *Integer Programming.* Ed. Addison-Wesley, USA.

[13] Weingartner, H. M., & Ness, D. N. (1967). *Methods for the solution of the multidimensional 0/1 knapsack problem.* Operations Research, 15(1), 83-103.

[14] Yamada, T., Kataoka, S., & Watanabe, K. (2002) *Heuristic and exact algorithms for the disjunctively constrained knapsack problem.* Information Processing Society of Japan Journal, 43(9).

Páginas de internet:

- Recuperado de `https://www.theorsociety.com`
- Recuperado de `http://hjemmesider.diku.dk/~pisinger/codes.html`
- Recuperado de `https://people.sc.fsu.edu/~jburkardt/datasets/knapsack_01/knapsack_01.html` Instancias del problema de la mochila para probar los heurísticos y metaheurísticos
- Recuperado de `www.transporte.mx/guia-de-compra-para-camionetas-de-carga.` Capacidad de las camionetas y tipo de camionetas.
- Recuperado de `http:www.campomexicano.gob.mx/portal_siap/Integracion/EstadisticaDerivada/InformaciondeMercados/Mercados/snim/dffa1101.htm`Pesos de verdura y fruta por caja
- Recuperado de `https://carplanet.mx/noticia/general/las-5-pickups-pequenas-mas-eficientes/55bc55c01bddf` Capacidad de camionetas chicas
- Recuperado de `https://www.ficeda.com.mx/content/boletines_mayoreo/BOLETIN_MAYOREO_2011_04_15.pdf` Pesos de las frutas incluyendo los de importación
- Recuperado de `http://www.campomexicano.gob.mx/mercados_nl/Presenta.phtml?central=150` Pesos de las frutas incluyendo los de importación

Software

- TeXShop Versión 4.76(4.76) 2022
- PowerPoint 2011
- GeoGebra Classic Versión 5 2018
- Student Lingo Versión 17 2017

Apéndice A

Glosario

- **Algoritmo:** Es un conjunto ordenado y finito de operaciones que permiten solucionar un problema en específico.
- **Algoritmo de tiempo polinomial:** Es el algoritmo cuyo número operaciones que se necesitan para aplicarlo se pueden acotar por medio de una función polinomial del tamaño del problema.
- **Algoritmo exacto:** Son algoritmos que proporcionan la solución óptima del problema de optimización (si existe), cuentan con una prueba de optimalidad.
- **Alternativas:** En un problema de programación dinámica son las posibles decisiones que se pueden tomar en cada etapa.
- **Clase NP:** Conjunto que contiene a los problemas para los cuales por medio de un algoritmo se puede verificar la veracidad de la respuesta en un tiempo polinomial.
- **Clase P:** Conjunto que contiene a los problemas de decisión que pueden ser resueltos por un algoritmo, el cual siempre finaliza con una respuesta (sí o no) en tiempo polinomial.
- **Combinación lineal convexa:** Sean x y y dos puntos, una combinación lineal convexa es $\alpha x + (1 - \alpha)y$ tal que $\alpha \in [0, 1]$
- **Combinación lineal convexa estricta:** Sean x y y dos puntos, una combinación lineal convexa es $\alpha x + (1 - \alpha)y$ tal que $\alpha \in (0, 1)$
- **Conjunto convexo:** Es el conjunto tal que si x y y son dos elementos del conjunto, entonces la combinación lineal convexa de dichos elementos está contenida en el conjunto.
- **Ecuaciones recursivas:** En un problema de programación dinámica son las ecuaciones que contienen al rendimiento acumulado.
- **Eficiencia de un algoritmo:** Porcentaje que indica que tan buena fue la solución obtenida por medio de un algoritmo.

- **Estado del sistema:** En un problema de programación dinámica proporciona la información necesaria para tomar una decisión en la etapa actual.
- **Etapas:** En un problema de programación dinámica son subproblemas que conforman al problema original.
- **Fenotipo:** Decodificación del genotipo para proporcionar la solución real, binaria o entera de un problema.
- **Función objetivo de un PPL:** Función lineal que se desea optimizar en un problema, las variables que la conforman son las variables de decisión.
- **Genotipo:** Arreglo binario de 8 bits que contiene la información de una solución de un problema codificada.
- **Instancia de un problema de optimización:** Es una pareja (F, c), donde: F es el conjunto de soluciones factibles.c es la función de costo. c.
- **Lista tabú:** Contiene a las soluciones que se han visitado recientemente y están prohibidas por cierto número de iteraciones.
- **Metaheurística:**Es un método que guía y modifica otras heurísticas para producir soluciones más allá de las que normalmente se generan en un heurístico de búsqueda.
- **Método Constructivo**: Construye paso a paso la solución del problema.
- **Métodos de búsqueda por entornos (por vecindades)**: Comienza con una solución inicial, ésta se modifica sucesivamente para obtener una solución final que es la mejor encontrada hasta ese momento.
- **Método de descomposición:** Descompone el problema original en problemas más sencillos de resolver,
- **Método de manipulación del modelo (relajación)**: Obtiene una solución del problema original a partir de un problema relajado
- **Método heurístico:** Es un procedimiento simple que está basado en el sentido común, éste proporciona una buena solución a problemas difíciles, de un modo fácil y rápido,
- **Operador de cruza:** Procedimiento que intercambia las colas de 2 genotipos para generar 2 genes nuevos, ésta puede ser por un punto o por dos puntos.
- **Operador de mutación:** Procedimiento que intercambia las entradas de un genotipo, ceros por unos y unos por ceros.
- **Problema combinatorio:** Es el problema cuyas instancias cuyos elementos de F son soluciones discretas.
- **Problema de decisión:** Problema que solo consta de una pregunta que puede ser contestada con un "si.º "no"

- **Problema de optimización:** Se compone del conjunto de todas las instancias del problema.
- **Problema de programación lineal (PPL):** Consta de una función lineal a optimizar sujeta a restricciones lineales.
- **Problema de programación lineal entera (PPLE):** Consta de una función lineal a optimizar sujeta a restricciones lineales, además, todas, alguna(s) de las variables del problema tienen la restricción de ser enteras .
- **Problema NP-completo:** Un problema es NP-completo si es NP y todos los problemas que son NP se reducen en tiempo polinomial a este.
- **Problema NP-duro:** Es el problema de optimización cuyo problema de decisión asociado a él es NP-completo.
- **Punto extremo:** Solución que no se puede expresar como combinación lineal convexa estricta de 2 elementos diferentes en F_P
- **Reducción polinomial:** Q_1 se reduce en tiempo polinomial a Q_2 si y sólo si se encuentra un algoritmo A_1 de tiempo polinomial que transforma el problema Q_1 a el problema Q_2.
- **Región factible:** Conjunto que contiene a todos las soluciones factibles, se denota por F_P
- **Relajación PL de un PPLE:** El el PPL que resulta al omitir las restricciones de que las variables deben ser enteras.
- **Restricciones de un PPL:** Conjunto de desigualdades o ecuaciones lineales cuyas variables son las variables de decisión.
- **Solución factible:** Punto que satisface todas las restricciones del problema de optimización.
- **Solución no factible:** Solución que no satisface al menos una restricción del problema.
- **Solución óptima:** Solución factible que optimiza a la función objetivo.
- **Tabla de frecuencia:** Es la tabla que lleva un conteo del número de veces que se ha visitado esa solución.
- **Variable binaria:** Variable que solo puede tomar el valor de 0 io1
- **Variables de decisión:** En un problema de programación dinámica son las variables que involucran a las alternativas en cada etapa.
- **Variables de estado:** En un problema de programación dinámica son las variables que nos proporcionan las condiciones que existen al empezar una etapa.

- **Variable entera:** Variable que solo puede tomar valores enteros.
- **Vecindad en torno a x:** Es el conjunto cuyos elementos son vecinos de x.
- **Vecino de x:** x' es un vecino de x si x' se puede obtener directamente a partir de x mediante una operación llamada movimiento, esta puede ser una permutación .

Apéndice B

Código en Python

Este apéndice contiene el código de los programas desarrollados en Python para el problema de la mochila binario.
Todos los códigos tienen los datos de una instancia, éstos son de la siguiente forma:

- $n = 5$
  Número de artículos

- $c = [24, 13, 23, 15, 16]$
  Beneficio por artículo incluido (coeficientes de la función objetivo

- $a = [12, 7, 11, 8, 9]$
  Peso unitario por artículo incluido

- $b = 26$
  Capacidad de la mochila

Si se desea cambiar de instancia, solo se deben capturar los datos con el mismo formato.

B.1. Código del algoritmo glotón

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Mon Nov 26 01:24:11 2018

@author: ANGA
"""

#Programa que resuelve el problema de la mochila
#utilizando un heurístico glotón
import math
import random
import numpy
n=5
c=[24,13,23,15,16]
a=[12,7,11,8,9]
b=26
op=0.00000000000001
m=[]
m1=[]
#Crea la lista de la utilidad marginal
for i in range (n):
    elemento=c[i]/a[i]
    m.insert(i,elemento)
for i in range (n):
    for j in range (n):
        if m[i]==m[j]:
            m[j]=m[j]+op
            op=op+0.00000000000001
            j=j+1
print("cocientes",m)
#for que ordena en forma acendente a la lista de utitidad marginal
for i in range (n):
    elemento=m[i]
    m1.insert(i,elemento)
m1.sort()
print("cocientes oredenados",m1)
m1.reverse()
print("cocientes oredenados",m1)
#ciclos que rompe empates arbitrariamente entre los cocientes
x=numpy.zeros((n))
i=0
j=0
for i in range(0,n):
    for j in range(0,n):
        if m1[i]==m[j]:
            # suma=suma+a[j]
            if b/a[j] >=1 and x[j]==0:
                x[j]=1
                b=b-a[j]
            elif b/a[j] <1 and x[j]==0:
                x[j]=0
                b=b
        else:
            j=j+1

print("solución óptima",x)
v_optima=0
for i in range (n):
    v_optima=v_optima+c[i]*x[i]
print("valor optimo",v_optima)
```

Figura B.1: *Código en Python del algoritmo Glotón*

Fuente: Elaboración propia utilizando PowerPoint (2010)

B.2. Código del algoritmo búsqueda en su entorno

```python
def evalua(n,a,c,X):
    b1=0
    v_optima=0
    for i in range (n):
        b1=b1+(a[i]*X[i])
    if b1<=b:
        for j in range (n):
            v_optima=v_optima+c[j]*X[j]
    elif b1>b:
        for j in range (n):
            v_optima=v_optima+c[j]*X[j]
        v_optima=v_optima-500
    return v_optima

def sol_inicial(n,a,c,b):
    m=[]
    m1=[]
    op=0.00000000000001
    #Crea la lista de la utilidad marginal
    for i in range (n):
        elemento=c[i]/a[i]
        m.insert(i,elemento)
    for i in range (n):
        for j in range (n):
            if m[i]==m[j]:
                m[j]=m[j]+op
                op=op+0.00000000000001
                j=j+1
    print("cocientes",m)
    #for que ordena en forma acendente a la lista de utitidad marginal
    for i in range (n):
        elemento=m[i]
        m1.insert(i,elemento)
    m1.sort()
    print("cocientes oredenados",m1)
    m1.reverse()
    print("cocientes oredenados",m1)
    #ciclos que rompe empates arbitrariamente entre los cocientes
    X=numpy.zeros((n))
    i=0
    j=0
    for i in range(0,n):
        for j in range(0,n):
            if m1[i]==m[j]:
                # suma=suma+a[j]
                if b/a[j] >=1 and X[j]==0:
                    X[j]=1
                    b=b-a[j]
                elif b/a[j] <1 and X[j]==0:
                    X[j]=0
                    b=b
            else:
                j=j+1
    return X

import math
import random
import numpy
n=7
c=[70,20,39,37,7,5,10]
a=[31,10,20,19,4,3,6]
b=50
X=numpy.zeros((n))
X1=numpy.zeros((n))
X2=numpy.zeros((n))
b1=0
b2=1
v_optima=0
v_optima1=0
v_optima2=0

X=sol_inicial(n,a,c,b)
v_optima=evalua(n,a,c,X)
v_optima1=v_optima
print("X inicial",X)
print("F objetivo inicial",v_optima1 )
#Termina ciclo que crea la solución inicial
#ciclo que crea el vecino de X
n_iteraciones=0
fobj=0
while n_iteraciones<50:
    a2=random.randrange(n)
    a2=int(a2)
    for i in range (0,n):
        X1[i]=X[i]
    if X[a2]==0:
        X[a2]=1
    elif X[a2]==1:
        X[a2]=0
    v_optima=evalua(n,a,c,X)
    v_optima2=v_optima
    print("X",X)
    print("F objetiv",v_optima2 )
    #Termina ciclo que crea al vecino
    #ciclo que cambia de vecino
    if v_optima2>v_optima1:
        v_optima1=v_optima2
    elif v_optima2<v_optima1:
        for i in range (0,n):
            X[i]=X1[i]
    n_iteraciones=n_iteraciones+1
print("X óptima",X)
print("F objetivo",v_optima1 )
```

Figura B.2: *Código en Python del algoritmo Búsqueda en su entorno*

Fuente: Elaboración propia utilizando PowerPoint (2010)

B.3. Código del algoritmo de Recocido Simulado

```python
#Función que evalua la solución guardada en X[]
def evalua(n,a,c,X):
    b1=0
    v_optima=0
    for i in range (n):
        b1=b1+(a[i]*X[i])
    if b1<=b:
        for j in range (n):
            v_optima=v_optima+c[j]*X[j]
    else:
        for j in range (n):
            v_optima=v_optima+c[j]*X[j]
        v_optima=v_optima-5000000
    return v_optima
import math
import random
import numpy
n=7
c=[70,20,39,37,7,5,10]
a=[31,10,20,19,4,3,6]
b=50

X=numpy.zeros((n))
X1=numpy.zeros((n))
X2=numpy.zeros((n))
b1=0
b2=1
tem=100
v_optima=0
v_optima1=0
v_optima2=0
#ciclo que crea una solución factible inicial aleatoriamente
while b2==1:
    b1=0
    X=numpy.zeros((n))
    for i in range (0,n):
        a1=random.random()
        if a1<=0.5:
            X[i]=0
        else:
            X[i]=1
    for i in range (n):
        b1=b1+(a[i]*X[i])
    if b1<=b:
        b2=2
    else:
        b2=1
    v_optima=evalua(n,a,c,X)
    v_optima1=v_optima

#Termina ciclo que crea la solución inicial
#ciclo que crea el vecino de X
n_iteraciones=0
fobj=0
print("Solución inicial",X)
print("F óptima",v_optima1)
while n_iteraciones<1000:
    a2=random.randrange(n)
    a2=int(a2)
    for i in range (0,n):
        X1[i]=X[i]
    if X[a2]==0:
        X[a2]=1
    elif X[a2]==1:
        X[a2]=0
    v_optima=evalua(n,a,c,X)
    v_optima2=v_optima
    #Termina ciclo que crea al vecino
    #ciclo que cambia de vecino
    if v_optima2>v_optima1:
        v_optima1=v_optima2
    elif v_optima2<v_optima1:
        deltaf=v_optima1-v_optima2
        prob=math.exp(-deltaf/tem)
        p=random.random()
        if p<=prob:
            v_optima1=v_optima2
        else:
            for i in range (0,n):
                X[i]=X1[i]
    tem=tem*0.9
    n_iteraciones=n_iteraciones+1
print("X óptima",X)
print("F objetivo",v_optima1 )
```

Figura B.3: *Código en Python del algoritmo de Recocido Simulado*

Fuente: Elaboración propia utilizando PowerPoint (2010)

B.4. Código del algoritmo Genético

```python
def evalua(n,a,X,c):
    import math
    import random
    import numpy
    b1=numpy.zeros(50)
    v_optima=[]
    for i in range (50):
        a2=0
        v_optima.insert(i,a2)
    for j in range (50):
        for i in range (n):
            val=a[i]*X[j][i]
            b1[j]=b1[j]+val
    for k in range(50):
        if b1[k]<=b:
            for j in range (n):
                v_optima[k]=v_optima[k]+c[j]*X[k][j]
        else:
            for j in range (n):
                v_optima[k]=v_optima[k]+c[j]*X[k][j]
            v_optima[k]=v_optima[k]-5000
    return v_optima
def selección(n,X,v_optima):
    num=0
    d=0
    Xnueva=numpy.zeros((50,n))
    v1_optima=[]
    for i in range (50):
        a2=0
        v1_optima.insert(i,a2)
    while d<45:
        for i in range(45):
            g=random.randrange(50)
            g=int(g)
            if v_optima[i]>=v_optima[g]:
                for j in range(n):
                    Xnueva[i][j]=X[i][j]
                    v1_optima[i]=v_optima[i]
                d=d+1
            else:

                for j in range(n):
                    Xnueva[i][j]=X[g][j]
                    v1_optima[i]=v_optima[g]
                d=d+1
    num=0
    while num<5:
       for i in range(5):
           M=max(v_optima)
           #print(M)
           p=v_optima.index(M)
           Xnueva[45+i]=X[p]
           v1_optima[45+i]=v_optima[p]
           v_optima[p]=0
           num=num+1
    return(Xnueva,v1_optima)
import math
import random
import numpy
n=8
c=[350,400,450,20,70,8,5,5]
a=[25,35,45,5,25,3,2,2]
b=104

X=numpy.zeros((50,n))
Xnueva=numpy.zeros((50,n))
for j in range (50):
    for i in range (n):
        a1=(random.random())
        #print(a)
        if a1<=0.5:
            X[j][i]=1
        else:
            X[j][i]=0
for j in range (50):
    for i in range (n):
        X[j][i]=int(X[j][i])
print("solución inicial")
print(X)
#print("f_inicial")
v_optima=evalua(n,a,X,c)
v_optima=evalua(n,a,X,c)
print(v_optima)
Xnueva,v1_optima=selección(n,X,v_optima)
print(Xnueva)
print(v1_optima)
num=0
while num<5:
    for i in range(45):
        probcruza=random.random()
        if probcruza<=0.8 and i<=48:
            r=random.randrange(n)
            r=int(r)
            for j in range(0,r):
                X[i][j]=Xnueva[i][j]
                X[i+1][j]=Xnueva[i+1][j]
            for j in range (r,n):
                X[i][j]=Xnueva[i+1][j]
                X[i+1][j]=Xnueva[i][j]
            probmuta=random.random()
            if probmuta<=0.01:
                q=random.randrange(n)
                for j in range (q,n):
                    if X[i][j]==0:
                        X[i][j]==1
                    elif X[i][j]==1:
                        X[i][j]==0
            i=i+1
        else:
            for j in range(0,n):
                X[i][j]=Xnueva[i][j]
    print("solución",num)
    print(X)
    print("f",num)
    v_optima=evalua(n,a,X,c)
    print(v_optima)
    Xnueva,v1_optima=selección(n,X,v_optima)
    print("solución nueva",num)
    print(Xnueva)
    print("f nueva",num)
    print(v1_optima)
    num=num+1
#Procedimiento que obtiene el óptimo después de num de iteraciones
optimo=max(v1_optima)
print("Optimo",max(v1_optima))
a5=v1_optima.index(optimo)
Xoptima=X[a5]
print("X=(",Xoptima,")")
```

Figura B.4: *Código en Python del algoritmo Genético*

Fuente: Elaboración propia utilizando PowerPoint (2010)