Universidad Michoacana de San Nicolás de Hidalgo
División de Posgrado de la Facultad de Ingeniería Eléctrica

DETERMINACIÓN DE LA INTERSECCIÓN ENTRE UNA HIPERCAJA Y UN HIPERPLANO BASADA EN LA MODIFICACIÓN AL MÉTODO SIMPLEX

TESIS

Que para obtener el grado de

MAESTRO EN CIENCIAS DE LA INGENIERÍA ELÉCTRICA

Presenta

Ing. Paulina Rodríguez Cárdenas

Dr. Jaime Cerda Jacobo

Director de Tesis

Dr. Juan José Flores Romero

Co-Director de Tesis

Morelia Michoacán, Julio 2016.

A mi padre Rafael Rodríguez Luna. †

A mi abuela Polly Lozano Benitez. †

Lista de Publicaciones

"A Non-basic Variable based approach to solve LPs"
Paulina Rodriguez Cárdenas, Jaime Cerda Jacobo, Jose Alberto Avalos.
prodriguez@dep.fie.umich.mx,jcerda@umich.mx, javalos@umich.mx
Graduate Studies Division, Electrical Engineering School
Universidad Michoacana de San Nicolas de Hidalgo
Publicado en 2015 IEEE International Autumn Meeting on Power, Electronics and Computing (ROPEC 2015) 978-1-4673-7121-6/15/31.00 c 2015 IEEE

Resumen

En este proyecto se presenta una aplicación que permite resolver el problema para determinar las intersecciones entre una hipercaja y un hiperplano; para ello se propone una implementación mediante herramientas de programación lineal, tomando como base el método simplex. Dicho método se modifica cambiando algunos de los principios por los que se rige el Simplex, como lo son el criterio de optimalidad, ganancia, factibilidad y el criterio de paro, de acuerdo a las necesidades particulares que se presentan en este trabajo el método solución llevara por nombre HHI (por sus siglas en inglés Hyperbox and Hyperplane Intersection). El objetivo final de HHI es determinar todos los vértices de intersección entre los dos cuerpos geométricos deacuerdo a los criterios modificados. Para lo anterior, el algoritmo propuesto se basa en un proceso de eliminación que se genera en base a un algoritmo recorrido de grafos a profundidad (DFS) sobre una busqueda ciega, por lo que es utilizado para trasladarse entre los vértices de intersección que estarán asociados a las soluciones básicas factibles del problema. Como primer paso la propuesta de este proyecto consiste en utilizar el método simplex (MSVNB), para llegar aun primer punto de intersección entre una hipercaja y un hiperplano para esto utiliza un tableau que contenga únicamente las variables no básicas en sus columnas, además de una columna unitaria auxiliar para realizar las operaciones. Esto implica que el número de operaciones que comúnmente se llevan a cabo en el método simplex disminuyan. Una vez obtenido el primer vértice de intersección entre la hipercaja y el hiperplano, se realiza una modificación en los criterios que rigen al método simplex, para que el algoritmo sea capaz de determinar los vértices de intersección entre la hipercaja e hiperplano, para que finalmente mediante un algortimo de recorrido de grafos y una estructuras de control basadas en tablas hash se pueda accesar a todos los vértices de intersección que hay en el sistema. Garantizando la obtención de todos vértices de intersección en el sistema evitando repeticion de vértices, ciclos infinitos y operaciones innecesarias.

Palabras Clave: Hipercaja, Hiperplano, Simplex, Programación Lineal, Intersección.

Abstract

In this project an application to solve the problem to determine the intersections between a hyperbox and a hyperplane is presented; to accomplish it, an implementation by linear programming tools, based on the simplex method is proposed. This a method is modified according to the particular needs that arise in this work, which we will call HHI (by its acronym in English Hyperbox and Hyperplane Intersection). The main goal of HHI is to determine all the vertexes of intersection between the two geometric bodies. For this, the proposed algorithm is based on a process of elimination that is generated based on a Depth First Search algorithm (DFS) used to move between the vertexes of intersection that are associated with the basic feasible solution of the problem. The proposal to use the simplex method algorithm HHI, consist only to intend to use a tableau containing only non-basic variables in its columns, plus an auxiliary unit to perform the operations column. This implies that the number of operations commonly carried out in the simplex method decrease. Once obtained the first vertex of an intersection between the hyperbox and the hyperplane, the algorithm will be able to find all the other vertexes of intersection between the hyperbox and hyperplane to make a change in the criteria of optimality of the simplex method, in addition to structures control to ensure that the vertexes of intersection are not visited for the second time, cycles and thus unnecessary operations are avoided.

Palabras Clave: Hyperbox, Hyperplane, Simplex, Lineal Programming, Intersection.

Contenido

Lista de Figuras

Lista de Tablas

Lista de Símbolos

Z	Funcion objetivo.
h	Hiperplano.
x	Conjunto de variables de desición.
c	Coeficientes de la función Objetivo.
b	Coeficientes del vector derecho de la ecuación RHS.
$\bar{x}$	Limite superior para el conjunto de variables de desición.
n	Conjunto de variables de desición deacuerdo al tamaño de la dimención de experimentación.
m	Conjunto de variables de holgura (restricciones).
N	Conjunto de lineas en el plano que se cruzan.
k	Número de aristas de la hipercaja.
A	Matriz de coeficientes.
$\mathcal{B}$	Conjunto de índices asociados a las columnas de las variables básicas.
$\mathcal{N}$	Conjunto de índices sociados a las columnas de las variables no básicas.
$x_{\mathcal{B}}$	representación a las variables básicas
$x_{\mathcal{N}}$	representación a las variables no básicas.
$c_{\mathcal{B}}$	representación a los coeficientes de las variables básicas.
$c_{\mathcal{N}}$	representación a los coeficientes de las variables no básicas.
$A_{\mathcal{B}}$	Submatriz básica.
$A_{\mathcal{N}}$	Submatriz no básica.
b	Lado derecho del vector de coeficientes.
$\mathbb{R}^m$	Espacio real de tamaño m.
$\mathbb{R}^n$	Espacio real de dimención n.
$\mathbb{R}^{n-m}$	Espacio real de tamaño $n-m$.
$\mathbb{R}^{m\times n-m}$	Espacio real de tamaño $m \times n - m$.
$\mathbb{R}^{m\times m}$	Espacio real de tamaño $m \times m$.
b_i	Instancia del conjunto de coeficientes del vector derecho de la ecuación RHS.
c_{ij}	Instancia del conjunto de coeficientes de la función Objetivo.
x_j	Instancia del conjunto de variables de desición.
B_f	El conjunto de soluciones básicas factibles.
B_n	El conjunto de soluciones básicas no factibles.
$\mathcal{P}$	Un problema de PL.
$\mathcal{P}_p$	Pólitope de $\mathcal{P}$.
B^{-1}	Submatriz básica inversa.
$\mathcal{K}$	columna auxiliar unitaria.
v	vértices de intersección adyacentes.

Capítulo 1

Introducción

La programación lineal (PL) es el campo de la optimización matemática dedicada a optimizar (maximizar o minimizar) una función lineal, denominada función objetivo Z, de tal forma que las variables de dicha función están sujetas a una serie de restricciones expresadas mediante un sistema de desigualdades también lineales. El método más usado para resolver problemas de programación lineal es el método simplex.

El presente estudio se enfoca en el método simplex basado en columnas no básicas (MSVNB) para desarrollar una herramienta para resolver el problema consistente en localizar los vértices que conforman la intersección entre una hipercaja y un hiperplano apoyándose en la programación lineal, ya que esta cuenta con herramientas que permiten describir y resolver el problema de la intersección a través de una serie de ecuaciones lineales tales que al aplicar el método simplex (MSVNB) se puedan determinar las soluciones del sistema.

En geometría, la intersección es el corte de dos cuerpos geométricos, que es respectivamente, un punto, una recta o una superficie. El caso más simple de una intersección en la geometría son los puntos comunes entre dos líneas, que puede ser un punto, sin embargo, en este estudio la intersección será más compleja.

La determinación de la intersección entre los distintos cuerpos geométricos es una tarea que le compete al área del álgebra lineal. En general la localización de la intersección se calcula a partir de un conjunto de ecuaciones lineales, las cuales pueden ser resueltas

numéricamente a través de un proceso de solución de ecuaciones lineales.

El problema de la intersección consiste en generar todos los vértices que, definen un politopo de la intersección entre el hiperplano y la hipercaja. Para lo anterior se pretende resolver este problema desde un enfoque de PL, donde se pueda describir la intersección entre la hipercaja e hiperplano en un sistema de ecuaciones lineales tal que se pueda aplicar una metodología basada en una variante del método simplex (MSVNB) en conjunto con una técnica de recorrido de grafos, que pueda asegurarse de encontrar todos los vértices del politopo y resolver todo el sistema.

El método simplex, que debe su origen a George B. Dantzig en 1947 [Dantzig55], es un procedimiento algebraico iterativo que tiene conceptos geométricos subyacentes, el cual recorre exclusivamente los vértices (o puntos extremos) de una región factible.

La región factible es determinada por un conjunto de desigualdades lineales (restricciones) generando un politopo en el cual se encuentran todos los puntos que satisfacen a todas las desigualdades a la vez, David G. Luenberger *et al.* [Luenberger89].

En este proyecto se propone un enfoque compuesto de varias etapas para determinar el conjunto de intersecciones, primero se aplica el método simplex (MSVNB) para encontrar un primer vértice de intersección, la segunda etapa consiste en que a partir de ese primer vértice de intersección se aplicará una modificación al método simplex (MSVNB) para determinar los vértices de intersección adyacentes, en la tercera etapa se incorpora un método de recorrido de grafos para poder visitar todos los vértices de intersección con el fin de no perderse entre las adyacencias de cada vértice de intersección evitando ciclos y repeticiones en los vértices visitados, y finalmente en se utilizan tablas de control basadas en tablas hash para tener un control de todos los vértices que se van procesando y tener toda la información de cada uno de ellos.

El problema de la intersección puede ser expresado equivalentemente como un problema de PL de la siguiente forma donde el hiperplano h es definido por $cx = b$:

$$\begin{aligned} \underset{x}{\text{Maximizar}} \quad & Z = cx \\ \text{Sujeto a} \quad & Ax \leq b \\ & 0 \leq x \leq \bar{x} \end{aligned} \tag{1.1}$$

Donde

$x \in \mathbb{R}^n$	Conjunto de variables de decisión
$\bar{x} \in \mathbb{R}^n$	Límite máximo
$c \in \mathbb{R}^n$	Coeficientes de la función objetivo
$b \in \mathbb{R}^m$	Coeficientes del vector derecho de la ecuación RHS(Right Hand Side)

Definiendo el problema de intersección como se muestra en la Fórmula (1.1), la función objetivo Z estará dada por el hiperplano y sujeta a las restricciones del sistema, que serán definidas por las propiedades de la hipercaja y el hiperplano, por lo tanto la región factible quedará acotada por el mismo hiperplano.

De tal manera que al describir el problema de la intersección como un problema de PL, se podrá resolver a través del método simplex modificado y las técnicas de recorridos de grafos, lo cual dará como resultado todos los vértices de la intersección entre los dos cuerpos geométricos, en el capítulo 4 se verá una explicación más detallada acerca del tema.

Al adecuar el método simplex MSVNB a las necesidades particulares abordadas en este proyecto, se le llamará al método simplex modificado *"El método de Intersección entre la Hipercaja e Hiperplano"*, o por sus siglas en inglés (HHI) Hyperbox and Hyperplane Intersection, el cual será el encargado de encontrar todos los vértices de intersección adyacentes que se encuentren en el sistema.

Históricamente, las ideas de programación lineal han inspirado varios de los conceptos centrales de la teoría de optimización tales como: la dualidad, la descomposición y la importancia de la convexidad y sus generalizaciones.

Algunos casos especiales de programación lineal, como los problemas de flujo de redes y flujo de mercancías, considerados en el desarrollo de las matemáticas, son suficientemente importantes para generar por sí mismos bastante investigación relativa a algoritmos

especializados en su solución; de la misma forma, hay algoritmos diseñados específicamente para casos en concreto, por ejemplo; los utilizados en la microeconomía y la administración de empresas, ya sea para aumentar al máximo los ingresos o reducir al mínimo los costos de un sistema de producción. Algunos ejemplos son la mezcla de alimentos, la gestión de inventarios, la gestión de recursos financieros, la asignación de recursos humanos, recursos de máquinas y la planificación de campañas de publicidad.

En las siguientes secciones se describe el planteamiento de la propuesta para resolver el problema de la intersección desde un enfoque de PL.

1.1. Planteamiento del Problema

El enfoque de la presente tesis es diseñar un algoritmo que permita resolver el problema de encontrar los vértices de intersección entre una hipercaja y un hiperplano, basándose en el método Simplex modificado que llamaremos HHI, enfocado desde el punto de vista de PL.

Existe una serie de conceptos geométricos que deben definirse antes de analizar la geometría de un problema de PL.

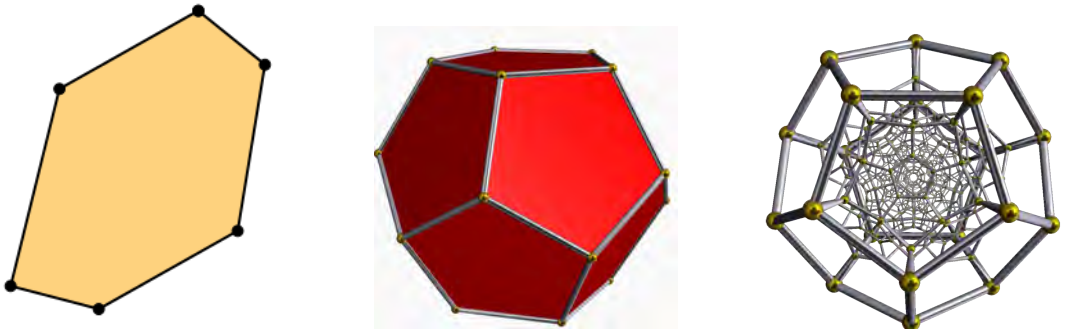
"Hipercaja": En geometría, una hipercaja es un análogo n-dimensional de un cuadrado (n = 2) y un cubo (n = 3). Se trata de una figura cerrada, compacta y convexa, cuyo esqueleto consiste en grupos de segmentos de líneas paralelas, opuestos, alineados en cada una de las dimensiones del espacio, perpendiculares entre sí y de distintas longitudes.

Ejemplo: una hipercaja de 0 dimensiones es un punto, una hipercaja de 1 dimensión es una arista, una hipercaja de 2 dimensiones se llama cuadrado, un hipercaja de 3 dimensiones se llama cubo, un hipercaja de 4 dimensiones se llama teseracto, etc.

"Hiperplano": En la geometría un hiperplano es un subespacio de una dimensión menor que su espacio ambiente. De ese modo si un espacio es tridimensional, entonces, sus hiperplanos son los planos de 2 dimensiones, mientras que si el espacio es bidimensional, sus hiperplanos serán en una dimensión. Esta noción se puede utilizar en cualquier espacio general en el que se define el concepto de la dimensión de un subespacio. En matemáticas un hiperplano está definido por el conjunto de puntos x_j que cumplen una ecuación cualquiera

del problema PL, como ya se definió anteriormente $h = \sum_j c_{ij}x_j = b_i$.

"Politopo" es la región convexa definida por la intersección de un conjunto finito de vértices, donde la región factible de un problema de programación lineal es parte del politopo, formando politopos de distintas formas, dependiendo de la dimensión con que se esté experimentando.



(a) ***Politopo en un espacio bidimensional.*** **(b)** ***Politopo en un espacio tridimensional.*** **(c)** ***Politopo en un espacio cuadrimensional.***

Figura 1.1: Figuras de politopos en distintas dimensiones.

Ahora bien, para desarrollar la aplicación que arroja los vértices de intersección que son soluciones al problema de la intersección entre dos cuerpos geométricos, se propone utilizar el algoritmo HHI, el cual a partir de un vértice de intersección entre la hipercaja y un hipercubo encuentra todos los vértices de intersección del sistema basandose en una modificación al método Simplex (MSVNB) en conjunto con una técnica de recorrido de grafos. La idea geométrica subyacente se describe a continuación.

El problema de la intersección entre una hipercaja y un hiperplano, se muestra en la Figura (1.2), donde se describe la geometría subyacente de la Ecuación (1.1).

La idea es que al iniciar la búsqueda de los vértices, primero se realiza la maximización de la función objetivo mediante el método Simplex (MSVNB), este comience a recorrer los vértices adyacentes dentro de la región factible, hasta que finalmente intersecte con uno de los bordes entre la hipercaja y el mismo hiperplano como se muestra en la Figura (1.3). El método Simplex (MSVNB) se utiliza para encontrar el primer vértice de intersección entre los dos cuerpos geométricos, el cual se explica a detalle en el capítulo 3.

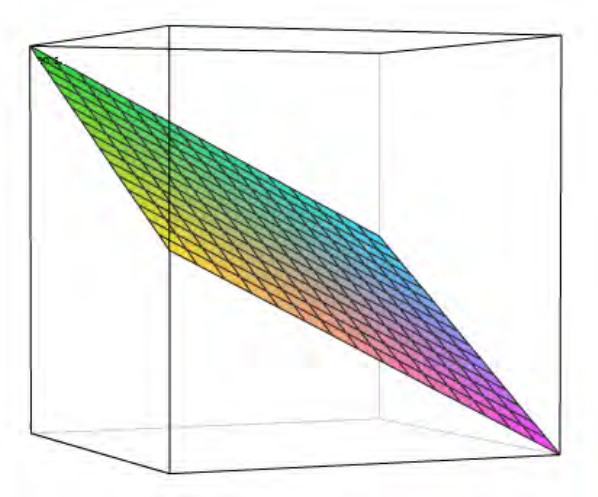


Figura 1.2: Intersección entre una hipercaja y un hiperplano.

Al partir de un vértice localizado en el origen, sobre una de las esquinas de la hipercaja, el método Simplex (MSVNB) se mueve entre los vértices adyacentes ubicados dentro de la región factible, mejorando la función objetivo Z entre cada movimiento, hasta que encuentra una solución básica factible (SBF) óptima. Esto se interpreta como haber encontrado el primer vértice solución de intersección entre la hipercaja y el hiperplano, pues como se mencionó anteriormente la función objetivo está dada por el hiperplano h y la región factible está acotada por el mismo hiperplano.

Todas las SBF están asociadas a un punto extremo o vértice y todo punto extremo del hiperplano h está asociado a alguna SBF. Las SBF a diferencia de las soluciones básicas se encuentran dentro de una región donde todas las restricciones del problema se cumplen, llamada comúnmente región factible.

El primer vértice de solución que devuelve el método Simplex (MSVNB), será el primer vértice de la intersección entre la hipercaja y el hiperplano. A partir de tal solución se pretende cambiar la dinámica del método Simplex (MSVNB) modificando algunos de los criterios por los que se rige, como lo son el criterio de optimalidad, la ganancia y el criterio de paro, así como realizar la integración de un algoritmo de eliminación de vértices basado en un recorrido de árboles en profundidad DFS, el cual se asegurará hacer posible el recorrido entre todos los vértices que comprenden la intersección.

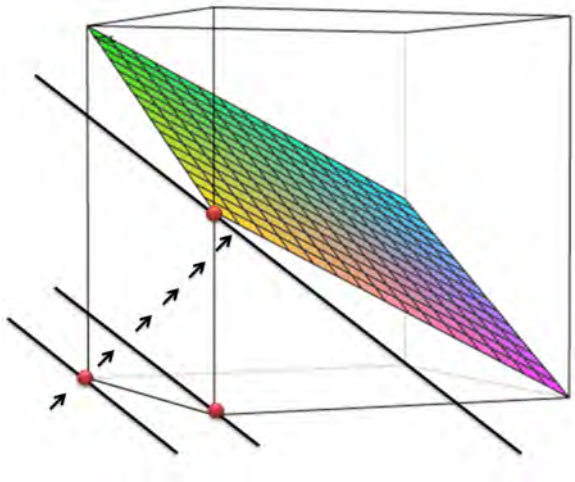


Figura 1.3: Proceso de maximización para encontrar los vértices de intersección solución del problema de la intersección entre una hipercaja y un hiperplano.

Los vértices de intersección solución entre la hipercaja e hiperplano son adyacentes, de tal forma que al hacer el recorrido entre los vértices se distingue un politopo convexo, donde cada vértice de intersección está interconectado con otro, así el algoritmo Simplex (MSVNB) se asegura que la primera solución encontrada esté interconectada con otros vértices de intersección. Por ejemplo, en la Figura (1.4) se puede observar una hipercaja de tres dimensiones cortada en distintas formas por un hiperplano de dos dimensiones. La forma que adopta el hiperplano dependerá de las restricciones y necesidades del problema que se desee resolver.

Como criterio aproximado se estima que el tiempo de ejecución del método Simplex es proporcional a un polinomio, en la variable x, donde x es el número de variables que describen al problema de PL, Bernard Kolman *et al.* [Bernard Kolman08].

Al tratarse de sistemas lineales se puede encontrar analíticamente el límite máximo del número total de vértices o soluciones básicas que existen en el sistema según la Fórmula del Binomial en (1.2). A partir de esta información se cuenta con un panorama del número total de vértices que se podrían verificar, sin embargo no todos los vértices que existen son soluciones al sistema, ya que únicamente son de interés los vértices de intersección que hay dentro de la región factible, que son SBFs y pueden evaluarse para ver cuál da un mejor

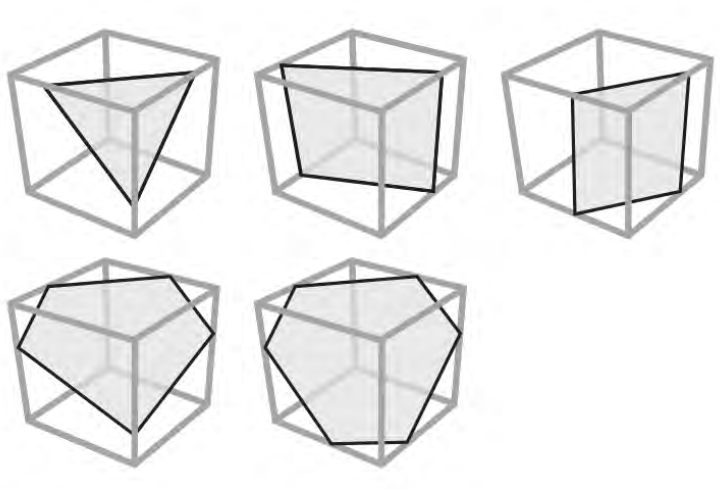


Figura 1.4: Intersección entre una hipercaja de tres dimensiones y un hiperplano de dos dimensiones por Christof Rezk Salama *et al.* [Rezk-Salama05]

valor en la función objetivo Z.

$$\binom{m+x}{m} \tag{1.2}$$

Donde

x Conjunto de variables de decisión

m Conjunto de variables de holgura (restricciones)

Los problemas reales suelen tener demasiadas variables, pueden llegar a tener cientos y hasta miles de variables sujetas a otras cientos o miles de restricciones. Por lo tanto, los problemas que se analizan son muy grandes, lo que hace necesario contar con una estrategia eficiente de búsqueda que permita encontrar únicamente aquéllos puntos que correspondan a una solución posible.

En tal razón, este proyecto propone el algoritmo HHI, ya que mejora la búsqueda de la intersección de una hipercaja con un hiperplano a través de las técnicas de PL.

1.2. Justificación

Las relaciones geométricas entre los objetos incluyen la determinación de si dos segmentos de línea se cruzan entre sí o sí un punto se encuentra por encima, por debajo o

sobre una línea dada.

Las relaciones de intersección entre varios objetos geométricos se calculan de manera eficiente, precisa y robusta, puede ser bastante difícil. Los cálculos de punto flotante son rápidos, pero adolecen de errores por redondeo, lo que puede dar lugar a decisiones erróneas. Estos errores pueden dar lugar a inconsistencias topológicas en representaciones de objeto, y las inconsistencias pueden causar los fallos en tiempo de ejecución. Algunos métodos utilizados para hacer frente a la robustez entre las relaciones geométricas discretas incluyen algoritmos de aproximación, que son robustos a errores de punto flotante, el cálculo de predicados geométricos exactamente usando aritmética adaptativa de punto flotante, aritmética exacta combinada con filtros rápidos de punto flotante y el diseño de algoritmos que se basan en un conjunto restringido relaciones entre los objetos geométricos, David M. Mount en [Mount97].

Las aplicaciones geométricas con frecuencia implican la búsqueda de la intersección de los objetos, por ejemplo: un polígono es simple de calcular si y sólo si dos de sus bordes se cruzan, un circuito impreso puede realizarse rápidamente sin cruces, siempre y cuando no haya dos conductores cruzándose, en gráficos de computadora un objeto que se desea mostrar oscurecerá otro si sus proyecciones sobre el plano de visión se intersectan. La importancia de algoritmos eficientes para estos problemas está creciendo cada vez más rápido; un solo circuito integrado puede contener decenas de miles de componentes, una escena complicada para la representación gráfica en computadora puede implicar un centenar de miles de vectores, y las bases de datos para el diseño arquitectónico debe ser capaz de almacenar más de un millón de elementos para estos fines, incluso los algoritmos cuadráticos son poco prácticos. Franco P. Preparata *et al.* [Franco P. Preparata85]

El tema común de los problemas anteriores es que todos ellos se pueden resolver rápidamente, si un algoritmo rápido está disponible para detectar si se cruzan cualquiera de los dos segmentos de N líneas en el plano.

Por lo que la primera parte de este trabajo se enfoca en utilizar el método Simplex (MSVNB) para encontrar la primera intersección entre una hipercaja y un hiperplano. Una vez que se tenga una SBF óptima o el primer vértice de intersección, se propone la solución de este problema encontrando todos los vértices de intersección a través del método

Simplex (MSVNB), mediante una modificación en los criterios de optimalidad y ganancia, la integración de un algoritmo basado en el recorrido de árboles DFS, el cual permite recorrer todos los vértices de intersección entre la hipercaja e hiperplano y así delimitar el politopo.

Los problemas de intersección desempeñan un papel fundamental en la geometría computacional, gráficos por computadora y la programación lineal. Todos estos problemas parecen estar ligados a la necesidad de requerir la detección de intersección mediante algún algoritmo rápido para que se pueda obtener la solución.

Stan Eisenstat *et al.* [Shamos76] fue el primero en sugerir que un algoritmo lineal para la intersección de dos polígonos convexos conduciría a un algoritmo $O(nlogn)$, en la intersección de semiplanos, y a partir de esta observación se hizo el punto de partida hacia muchos trabajos relacionados al problema de la intersección entre cuerpos geométricos.

1.3. Objetivos de la tesis

1.3.1. Objetivo general

El objetivo general de este proyecto radica en desarrollar una metodología basada en herramientas de programación lineal, específicamente en una modificación al método Simplex (MSVNB), con la finalidad de determinar los vértices de intersección entre una hipercaja y un hiperplano.

1.3.2. Objetivos particulares

- Convertir el problema de encontrar las intersecciones entre una hipercaja y un hiperplano en un problema de programación lineal.

- Implementar una modificación al método Simplex (MSVNB), para la búsqueda de los vértices de intersecciones entre la hipercaja y el hiperplano.

- Desarrollar un algoritmo de eliminación que se genera en base a un recorrido DFS, utilizado para trasladarse entre los vértices de intersección entre la hipercaja e hiperplano, los cuales están asociados a las SBFs del problema, además de implementar

tablas hash para generar las listas de vértices que son soluciones y llevar el control de los mismos.

1.4. Descripción de capítulos

Este Capítulo introduce al lector al problema de la intersección entre una hipercaja y un hiperplano. Así mismo se describen objetivos del presente trabajo y sus contenidos. El resto del documento guarda la estructura presentada a continuación.

En el Capítulo 2, se presentará el trabajo relacionado a este proyecto, así como los conceptos y aplicaciones teóricas de investigaciones anteriores. En el Capítulo 3, se explicará la implementación del método Simplex basado en columnas no básicas (MSVNB) y una descripción detallada del funcionamiento. El Capítulo 4, presentará cómo convertir el problema de la intersección en un problema de PL, después analizará el proceso para encontrar las intersecciones entre la hipercaja e hiperplano basadas en una modificación al método Simplex (MSVNB). También se explicará cómo se lleva a cabo el recorrido entre todos los vértices de intersección de acuerdo a las técnicas basadas en DFS y finalmente se describen las estructuras de control basadas en tablas hash. En el Capítulo 5, se explicará a detalle la implementación del algoritmo de solución HHI y la ventaja de aplicar esta técnica en específico en la búsqueda de la solución a la intersección de una hipercaja y un hiperplano. En el Capítulo 6, se mostrarán los resultados obtenidos en las pruebas para verificar y medir la calidad del algoritmo implementado. Finalmente, en el Capítulo 7, se presentarán las conclusiones y se enumerarán los aspectos sobre los que se puede mejorar el algoritmo.

Capítulo 2

Trabajo relacionado

En este capítulo se presentan los conceptos y aplicaciones teóricas de investigaciones anteriores relacionadas a este proyecto. En principio se analizarán las investigaciones donde se presentan metodologías para la solución del problema de la intersección. Posteriormente se presentarán datos expuestos en otras investigaciones previas que aportan información complementaria para la solución propuesta en este trabajo.

2.1. Aplicaciones de la intersección

En este capítulo se presentan conceptos teóricos de las herramientas utilizadas para el proyecto. Como ya se mencionó en el Capítulo 1, en este estudio se desarrolla un algoritmo para obtener vértices de intersección entre un hiperplano y una hipercaja, con la finalidad de localizar a los vértices de intersección y delimitar una región donde se cumplan con las restricciones (desigualdades) dadas por el sistema.

Para comenzar a resolver el problema de la intersección entre dos cuerpos geométricos, se debe tener en consideración que, localizar el lugar donde dos objetos geométricos se intersectan y computar la región de intersección, son problemas fundamentales en la geometría computacional.

Los problemas de la intersección geométrica pueden ser encontrados en un gran número de aplicaciones tales como son: embalaje geométrico y cubierta, componentes de diseño en VLSI, un mapa de superposición en los sistemas de información geográfica, la

planificación de movimiento y detección de colisiones. En el modelado de sólidos, el cálculo del volumen de intersección de las dos formas, es un paso importante en la definición de sólidos complejos. En gráficos de computadora, la detección de los objetos que se superponen en una ventana de visualización es un ejemplo de un problema de intersección asi como también detectar las intersecciones en una colección de sólidos geométricos. David M. Mount en [Mount97].

ning complex solids. In computer graphics, detecting the objects that overlap a viewing window is an example of an intersection problem, as is computing the rst intersection of a ray and a collection of geometric solids.(GEOMETRIC INTERSECTION David M. Mount)

Existen algunas técnicas para resolver el problema de la intersección entre una hipercaja y un hiperplano, a continuación se describen algunas de ellas.

2.2. Estado del arte

Algunos trabajos importantes están relacionados bajo este estudio, utilizan distintos enfoques, enseguida se mencionan.

El trabajo desarrollado por Carlos Lara *et al.* [Lara09] procesan los vértices de una hipercaja en lugar de sus aristas. Por definición, se sabe que existen 2^n vértices, entonces son de interés las propiedades de subconjuntos de vértices que evitan la complejidad de tiempo. La representación de conjuntos de vértices que se utilizan se relaciona con el concepto de *"squashed cubes "* introducido por Graham and Pollak *et al.* [Graham72].

Ahora bien, la investigación desarrollada por Rezk Salama *et al.* [Rezk-Salama05] se aplica un "Naive Algorithm", para encontrar los puntos de intersección entre dos cuerpos geométricos tomando a cada arista del sistema y procesandola para encontrar el punto donde se intersecta con el hiperplano. Cabe mencionar que este trabajo se avoca hacia el problema de determinar la localización exacta de todas las esquinas de intersección en una hipercaja y un hiperplano únicamente de tres dimensiones.

En el documento Calos Lara et al. [Lara09] propone implementar un algoritmo de manera sistemática para eficientar y generar nodos en los bordes; dado un nodo fron-

tera, se exploran todos los vértices no revisados para determinar posibles intersecciones en cada una de ellas. Esta exploración sistemática permite enfocarse en la región fronteriza, descartando las dos regiones antes y después del plano. La poda de esas regiones produce un costo computacional lineal en el número de vértices del hiperpolígono que representa la intersección.

En el documento Rezk Salama *et al.* [Rezk-Salama05], proveen un algoritmo de solución basado en el "Naive Algorithm" computando la intersección de un cubo con un plano en 3D. Su motivación conduce al campo de los gráficos a computadora y renderizado 3D, dado que sus aplicaciones tratan solo con objetos 3D, su trabajo no se extendió a las intersecciones en n dimensiones. En lugar de proporcionar un algoritmo general, los autores analizaron todos los escenarios posibles para el caso 3D.

Respecto al problema general consistente en determinar la posición de todas las esquinas de la intersección de una hipercaja con un hiperplano, no se encontraron más investigaciones que se enfocarán en este tema en particular.

Cabe señalar que tampoco se encontraron propuestas en la cual se expusiera el problema de intersección entre una hipercaja y un hiperplano desde el punto de vista de PL. Sin embargo, se encontraron datos relacionados al tema que pueden aportar valiosa información. A continuación se presentan cada uno de ellos.

Michael L. Gargano *et al.* [Gargano03], [Klavžar] proporcionó datos interesantes para calcular el número de bordes que contiene una hipercaja, de dos distintas maneras, como se muestra en la Fórmula (2.1) que se presenta enseguida:

$$n2^{n-1} = \sum_{k=1}^{n} k\binom{n}{k} \tag{2.1}$$

En las investigaciones que realizó Patrick E. O'Neil en [O'neil71] propuso originalmente dos problemas: ¿cuál es el número máximo de k bordes de una hipercaja que puede cortar un hiperplano? y ¿cuál es el número mínimo de h hiperplanos que cortan toda las aristas de una hipercaja? Utilizando la generalización de Baker y el lema de Sperner, O'Neil muestra un teorema donde establece que el número máximo de aristas k de una hipercaja pueden ser cortadas por un hiperplano está dada por la Fórmula (2.2).

$$k = \left(n - \left[\frac{1}{2}n\right]\right)\binom{n}{\frac{n}{2}} \tag{2.2}$$

Aunado a este trabajo, el matemático R. Ahlswede *et al.* [Ahlswede90] hicieron la propuesta del siguiente teorema donde explican que el máximo número de intersecciones que pueden encontrarse en entre una hipercaja y un hiperplano es de:

$$p\binom{n}{p} \tag{2.3}$$

Donde

$$p = \left\lceil \frac{n+1}{2} \right\rceil \tag{2.4}$$

Sustituyendo el valor de p en la Fórmula (2.3) quedaría como:

$$\left\lceil \frac{n+1}{2} \right\rceil \binom{n}{\left\lceil \frac{n+1}{2} \right\rceil} \tag{2.5}$$

n es la dimensión de experimentación

Si se grafica el comportamiento de las Fórmulas (2.2) y (2.5) se obtiene la Figura (2.1), donde se puede observar que son muy similares, las diferencias que presentan entre ellas son mínimas y pueden ser despreciables; se puede concluir que cualquiera de las dos representaciones dará el número de intersecciones máximos que puede encontrarse entre una hipercaja y un hiperplano.

Con estos datos se tiene una referencia más clara del posible número máximo de vértices de intersección a encontrar entre la hipercaja e hiperplano.

Ahora bien, se compara gráficamente la Fórmula (2.5) de R. Ahlswede contra el número de aristas que presenta la hipercaja, basada en la Fórmula (2.1) se obtiene la Figura (2.2). La cual muestra que el mayor número de intersecciones que puede tener una hipercaja en n dimensiones, siempre es menor al número de aristas, esto quiere decir que no existe ningún hiperplano que pueda cortar todas las aristas de una hipercaja.

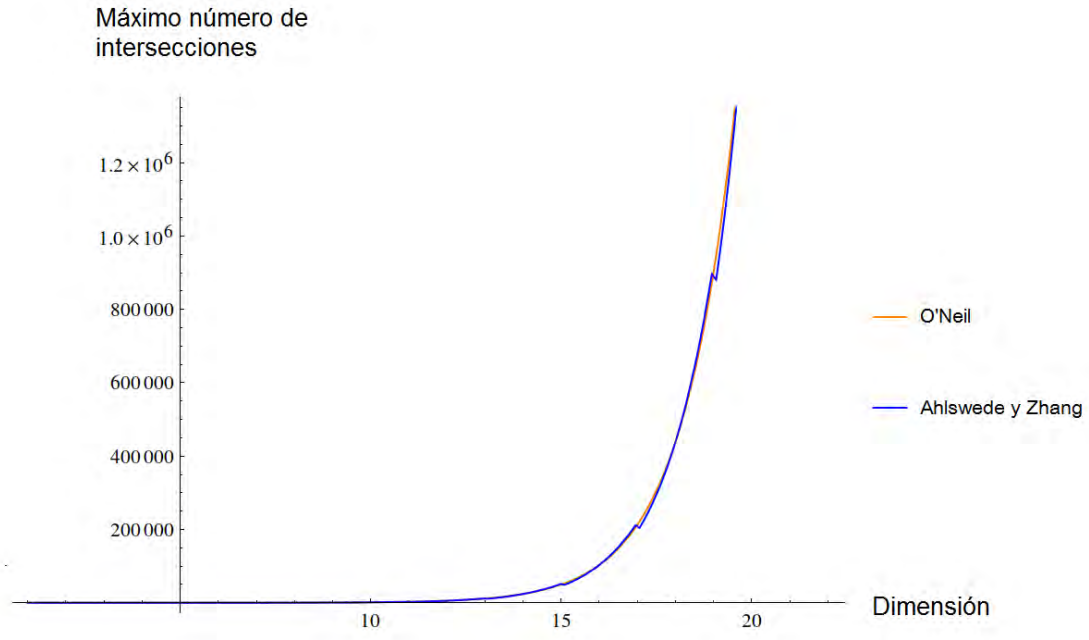


Figura 2.1: Comportamiento del número de intersecciones desde $n = 3$ hasta $n = 20$ dimensiones.

En cuanto a la segunda propuesta de O'Neil, a partir de la Fórmula (2.2) y la fórmula de Stirling se propone originalmente que el número mínimo de hiperplanos h necesarios para que se realice la intersección en todas las aristas de una hipercaja de dimensión n es $h = n$, O'Neil menciona que para una hipercaja de 3 dimensiones $h(3) = 3$.

La información relacionada con la finalidad de tratar de encontrar el número mínimo de hiperplanos requeridos para cortar todas las aristas de una hipercaja, se discute también Grünbaum *et al.* [Grünbaum73] extiende el trabajo de O'Neil tratando de generalizar su propuesta para todos los poliedros convexos.

Después Sohler *et al.* [Sohler00] ampliaó en gran medida la propuesta de O'Neil e incluso son capaces de proporcionar un límite inferior al propuesto por éste, en número mínimo de hiperplanos necesarios para cortar todos las aristas de una hipercaja de hasta siete dimensiones.

También Neil Calkin *et al.* [Calkin08] se aborda el problema de encontrar el número mínimo de hiperplanos necesarios que puedan cortar todas las aristas de una hipercaja

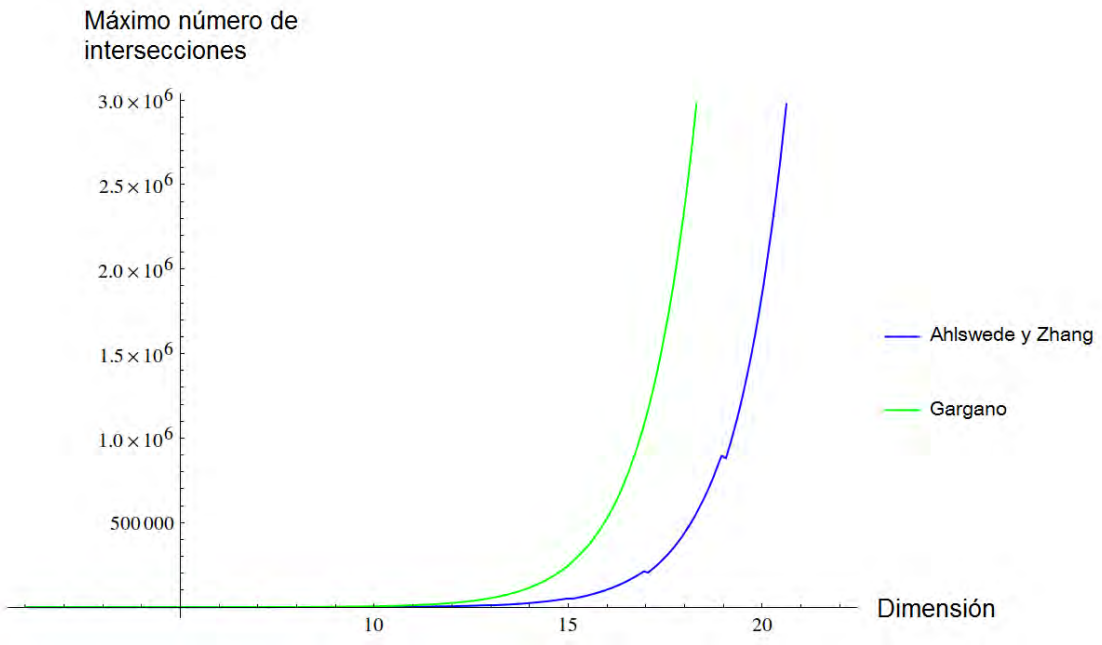


Figura 2.2: Número de intersecciones máximo vs número de aristas.

desde una perspectiva probabilística. En este trabajo se presenta el número de hiperplanos requeridos para cortar un borde con alta probabilidad y el número de hiperplanos requeridos para cortar todos los bordes de una hipercaja con alta probabilidad.

Otro dato importante es el número mínimo de intersecciones que se pueden obtener entre una hipercaja y un hiperplano, la cota mínima es igual al número de dimensiones siempre que el hiperplano h se encuentre a 45 grados de inclinación respecto de la hipercaja, si la hipercaja es de dimensión $n = 3$ el hiperplano h intersecta al menos tres veces con la hipercaja.

Así en la Figura (2.3), se observa que el número de intersecciones solución que se pueden obtener, siempre oscila entre el menor número de intersecciones y la cota máxima proporcionada por la Fórmula (2.5). En la cual nunca se obtiene un número de intersecciones mayor a la Fórmula (2.5).

Por otro lado se puede observar, el número de vértices de la hipercaja a partir de la séptima dimensión es mucho menor a la cota máxima de intersecciones que se pueden obtener y al número de aristas de la hipercaja como se muestra en la Figura (2.4).

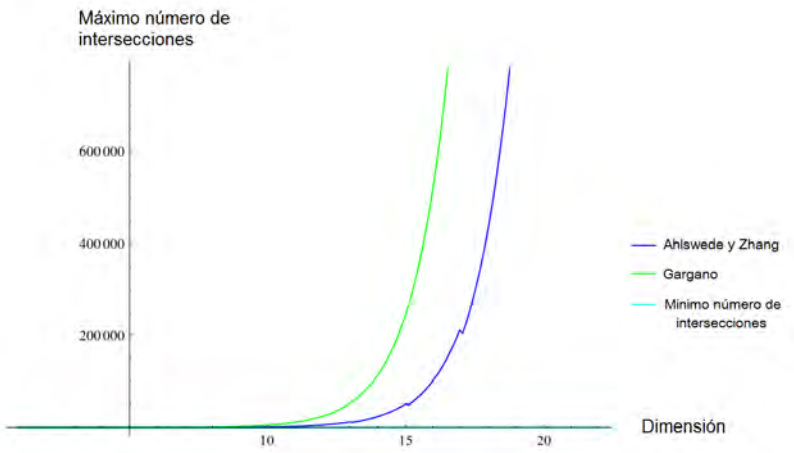


Figura 2.3: Número de aristas vs máximo número de intersecciones solución vs mínimo número de intersecciones solución.

Estos datos ayudan a identificar el número máximo posible de intersecciones entre una hipercaja y un hiperplano de acuerdo a las Fórmulas (2.2) y (2.5) dadas por Patrick E. O'Neil y R. Ahlswede respectivamente, conforme a las características del sistema, además de que se comprueba que el número de intersecciones solución no puede ser mayor al número de aristas de la hipercaja pero si puede llegar a ser mayor al número de vértices. Por último se puede observar que al aumentar el tamaño de las dimensiones, aumenta significativamente el número máximo de intersecciones solución que se encontrarán en el sistema. Con todos estos conceptos en los siguientes capítulos se procederá a explicar el diseño y la implementación del algoritmo propuesto.

2.3. Comentarios finales

En este capítulo se presentaron los fundamentos teóricos de las investigaciones enfocadas en el problema de la intersección entre una hipercaja y un hiperplano; además se brindó una explicación de los conceptos relevantes como el número máximo de vértices intersecciones que se pueden obtener entre una hipercaja y un hiperplano, también se analizó la relación existente entre el número de aristas y el número de nodos de la hipercaja respecto a la cota máxima de vértices de intersección siendo todo lo anterior necesario para

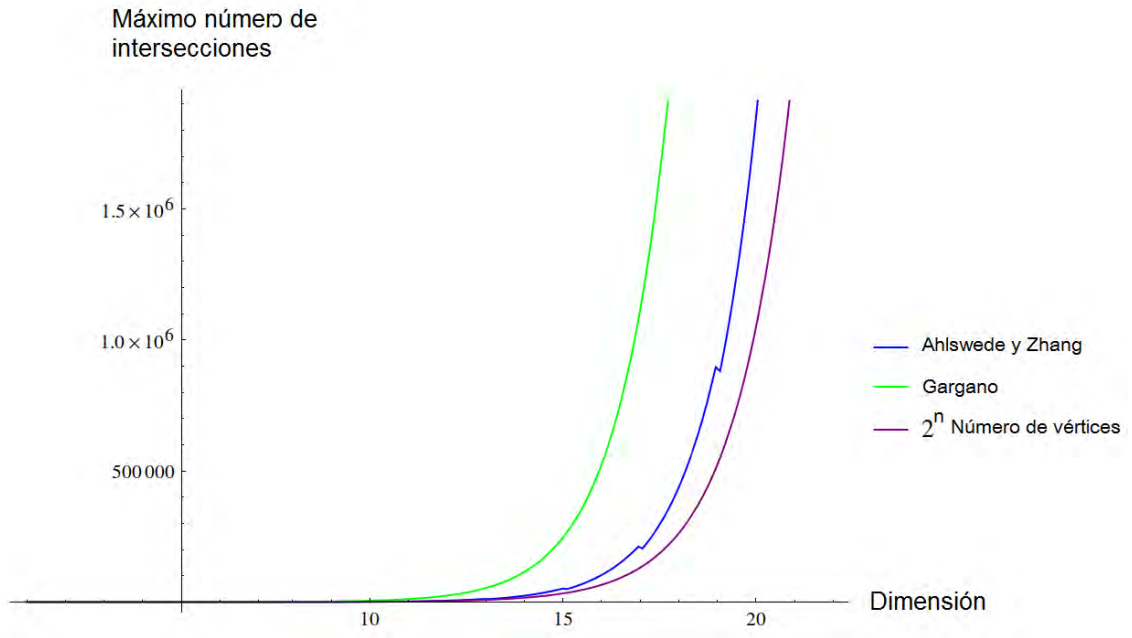


Figura 2.4: Máximo número de intersecciones vs número de aristas vs número de vértices.

entender el comportamiento del algoritmo propuesto en este proyecto. Una vez que han sido introducidos los conceptos, se procede a la explicación sobre el diseño y la implementación de la primera parte del algoritmo solución presentado, enfocado en el método Simplex (MSVNB) basado en columnas no básicas, con la finalidad de encontrar la primer intersección entre una hipercaja y un hiperplano. Este es el enfoque del próximo capítulo donde se brindará una descripción a detalle.

Capítulo 3

El método simplex basado en columnas no básicas

En este capítulo se describen algunas de las decisiones más importantes de diseño e implementación del método Simplex basado en columnas no básicas (MSVNB) el cual se utilizará para encontrar el primer vértice de intersección y se brinda una descripción del funcionamiento general del mismo.

3.1. Introducción

El método Simplex es el método más usado para resolver problemas de PL. Con él se busca optimizar una función lineal que involucra un conjunto de variables, sujetas a un conjunto de ecuaciones lineales, David Luenberger *et al.* [G.Luenberger08] y Klee.V *et al.* [Klee V72].

En este documento se asume, sin pérdida de generalidad, que el problema a optimizar es de maximización. El método Simplex es un método iterativo y permite ir mejorando la solución en cada iteración. El poliedro formado por la intersección de las restricciones, únicamente en la región factible, es llamado polítope. El hecho de que la función objetivo mejore en cada iteración, radica en que el método se mueve de un vértice del polítope a otro, siempre y cuando la función objetivo en el nuevo vértice sea mayor que en el anterior.

Dado que el número de vértices del polítope es finito, por consecuencia, siempre se hallará la solución.

El método del Simplex consiste en una serie de transformaciones lineales a lo largo de las iteraciones, en cada una de las cuales se va modificando la base del sistema, una variable a la vez; ello se realiza mediante una sucesión de tableaus [1] al ir cambiando de base. Teniendo en cuenta que muchos de estos cálculos no hacen falta para la determinación de cada nueva base, existen versiones alternas al Simplex (MSVNB), denominada Simplex matricial. Esta versión trata el problema explícitamente con matrices, pero siempre aparece la inversa de la matriz base, la cual produce una disminución de la eficiencia del método. Para revertir este efecto, anteriormente se han propuesto versiones al mismo, llamado Simplex revisado, para evitar el uso de la inversa, se propone la solución del sistema matricial como una serie de productos de factores de matrices elementales. En contexto, se utiliza el método Simplex (MSVNB) consistente en hacer un tableau que contenga únicamente columnas de las variables no básicas y una columna auxiliar unitaria donde se realizarán los cálculos, con la finalidad de relaizar un menor número de operaciones lo que significa almacenar un menor espacio de memoria en comparación con otras variantes del método Simplex, y de esta manera encontrar el primer vértice de intersección entre la hipercaja y el hiperplano.

3.2. El método Simplex

El método Simplex, es uno de los más utilizados para resolver problemas de PL. Es un algoritmo que determina los valores para un conjunto de variables no negativas, optimiza una función lineal sujeta a un cierto número de restricciones lineales, menciona Dantzig, G. B. en [Dantzig55, Dantzig63].

En este estudio, se tocarán los dos algoritmos principales usados para la solución del modelo de PL. El primero está basado en la formación de un tableau diseñado para resolver problemas de PL, haciendo operaciones en cada una de las columnas del tableau; el segundo, el Simplex matricial, ya no trabaja con el tableau, sino que, se basa en una formulación matricial para hacer las operaciones e ir determinando la solución óptima.

[1] Se utiliza el término "tableau"para referirse a la tabla matricial utilizada en el método Simplex Barry Render *et al.* [Render06]

El problema de PL en su forma estándar se define formalmente en las Ecuaciones (3.1).

$$\begin{aligned} \underset{x}{\text{Maximizar}} \quad & f = cx \\ \text{Sujeto a} \quad & Ax \leq b \\ & x \geq 0 \end{aligned} \tag{3.1}$$

Donde:

$x \in \mathbb{R}^n$	Conjunto de variables de desición
$c \in \mathbb{R}^n$	Coeficientes de la función objetivo
$A \in \mathbb{R}^{m \times n}$	Matriz de coeficientes
$b \in \mathbb{R}^m$	Coeficientes del vector derecho de la desigualdad

Considere el problema de la Tabla 3.1 el cual se presenta en su forma normal. El cual describe un sistema con una función objetivo Z sujeta a tres restricciones de desigualdad y dos variables de desición x_1 y x_2.

Tabla 3.1: Problema de PL en su forma normal.

Normal
Maximize $Z = 3x_1 + 2x_2$
s.t. $5x_1 + 10x_2 \leq 30$
$10x_1 + 5x_2 \leq 30$
$x_1 + x_2 \leq 3.5$

Para la aplicación del método Simplex se transforma el modelo de programación normal, formado por restricciones funcionales de desigualdad o igualdad, en un modelo de forma normalizado, integrado por restricciones de igualdad equivalentes. Esta conversión se logra con la introducción de variables de holguras[2] o también llamadas variables slacks.

[2]Variables de holgura o excedente. Son variables que se agregan a la restricción para que la relación de la restricción sea de igualdad (representa el valor que le hace falta al lado izquierdo para ser igual al lado derecho). Ambos tipos de variables tienen que cumplir con la restricción de no negatividad.

Entonces el ejemplo de la Tabla 3.1 queda como se presenta en la Tabla 3.2, en donde las variables de holgura seran descritas por s_1, s_2 y s_3.

Tabla 3.2: Problema propuesto de programación lineal.

Normalizado	Normalizado
Maximizar $z = 3x_1 + 2x_2$	Maximizar $z = 3x_1 + 2x_2 + 0s_1 + 0s_2 + 0s_3$
$5x_1 + 10x_2 + s_1 = 30$	$5x_1 + 10x_2 + 1s_1 + 0s_2 + 0s_3 = 30$
$10x_1 + 5x_2 + s_2 = 30$	$10x_1 + 5x_2 + 0s_1 + 1s_2 + 0s_3 = 30$
$x_1 + x_2 + s_3 = 3.5$	$1x_1 + 1x_2 + 0s_1 + 0s_2 + 1s_3 = 3.5$

De esta manera podemos apreciar una matriz identidad, formada por las variables de holgura las cuales solo tienen coeficiente 1 en su respectiva ecuación, por el ejemplo la variable de holgura s_1 solo tiene coeficiente 1 en la restricción correspondiente a una ecuación.

Siguiendo el protocolo del método Simplex una vez que el problema esta en su forma normalizada, se hace uso del Tableau sobre el cual se llevaran a cabo todas las operaciones matriciales. Entonces el método Simplex parte de una solución básica [3] inicial para realizar todas sus iteraciones, dado que todas las variables de la forma normalizada pueden ser elegidas para variables ser no básicas[4] (cero), en el tableau inicial las nuevas variables introducidas en la forma normalizada, son básicas (diferentes de cero), dado que su valor puede ser calculado trivialmente ($x_s i = b_j$), para más información acerca de las condiciones iniciales y conceptos del método simplex Vasek Chvatal en [Chvatal83].

Entonces el Tableau inicial quedaria como la Tabla 3.3 donde en las columnas aparecerán todas las variables del problema tanto de desición como de holgura. En las filas apareceran los coeficientes de las igualdades obtenidas, una fila para cada restricción. Donde c_j representa a los coeficientes de la función objetivo Z, c_N a los coeficientes de las variables no básicas, x_N a las variables no básicas las cuales estaran coloreadas en rojo, c_B a los coeficientes de las variables básicas, x_B a las variables básicas las cuales estaran coloreadas en verde, b a los coeficientes del lado derecho de las desigualdades, z_j En esta fila se consigna

[3] A variable in the basic solution (value is not 0). Vasek Chvatal en [Chvatal83]
[4] A variable not in the basic solution (value = 0). Vasek Chvatal en [Chvatal83]

la contribución total, es decir la suma de los productos entre término y c_b.

Tabla 3.3: Problema de PL, Tableau inicial

			c_j	3	2	0	0	0	
				Variables de desición		Variables de Holgura			b
c_N	x_N	c_B	x_B	x_1	x_2	s_1	s_2	s_3	
0	x_1	0	s_1	5	10	1	0	0	30
0	x_2	0	s_2	10	5	0	1	0	30
		0	s_3	1	1	0	0	1	3.5
			z_j	0	0	0	0	0	0
			$c_j - z_j$	3	2	0	0	0	0

Por lo que esta agrupación en las columnas del tableau con las variables de desición y las variables de holgura pude agruparse de manera distinta, agrupando ahora en terminos de las variables básicas y no básicas, describiendose a continuación.

Sea $\mathcal{B}$ el conjunto de índices asociados a las columnas básicas del sistema y $\mathcal{N}$ el conjunto de índices asociados a las columnas no básicas del sistema, en cualquier vértice del politope, las variables pueden dividirse en dos conjuntos: $x_{\mathcal{B}}$ representa a las variables básicas y $x_{\mathcal{N}}$ las variables no básicas, del mismo modo, los coeficientes de la función objetivo se particionan como $c_{\mathcal{B}}$ los coeficientes de las variables básicas y $c_{\mathcal{N}}$ los coeficientes de las variables no básicas y finalmente, la matriz A puede particionarse como $A_{\mathcal{B}}$ las columnas básicas del sistema y $A_{\mathcal{N}}$, las columnas no básicas.

Entonces, el problema original de las Ecuaciones (3.1), queda de la siguiente manera:

$$\begin{aligned} \underset{x}{\text{Maximizar}} \quad & f = c_N x_N + c_B x_B \\ \text{Sujeto a} \quad & A_{\mathcal{N}} x_N + A_{\mathcal{B}} x_B \leq b \\ & x_N \geq 0 \\ & x_B \geq 0 \end{aligned} \tag{3.2}$$

Donde:

$x \in \mathbb{R}^n$	Conjunto de variables
$\mathcal{B}$	Conjunto de índices de las variables básicas
$\mathcal{N}$	Conjunto de índices de las variables no básicas
$A_{\mathcal{B}} \in \mathbb{R}^{m \times m}$	Submatriz básica
$A_{\mathcal{N}} \in \mathbb{R}^{m \times n-m}$	Submatriz no básica
$c_{\mathcal{B}} \in \mathbb{R}^m$	Conjunto de variables básicas
$c_{\mathcal{N}} \in \mathbb{R}^{n-m}$	Conjunto de variables no básicas
$b \in \mathbb{R}^m$	Lado derecho del vector de coeficientes

Se observa como en la Tabla (3.4) se forma el tableau matricial del método Simplex.

Tabla 3.4: Tableau en forma matricial del método Simplex

	$\mathcal{N}$	$\mathcal{B}$	$\mathcal{RHS}$
1 ⋮ m	$A_{\mathcal{N}}$	$A_{\mathcal{B}}$	b
0	c_N	c_B	$-f$

Más adelante se descrirán algunos aspectos más relevantes de la versión de tableau, el método matricial, así como de la versión basada en columnas no básicas.

3.2.1. Simplex basado en Tableaus

El método Simplex básico o Simplex basado en tableaus, permite ir mejorando la solución en cada paso del procedimiento, comenzando con una solución básica y modificándola a lo largo de una variable como lo menciona Hélcio Vieira *et al.* [Vieira04], a través de un pivoteo parcial. Es decir, al intercambiar una variable básica y una variable no básica, siempre aumenta la utilidad o reduce el costo, hasta encontrar una solución óptima.

A continuación se explicarán a grandes rasgos las diferencias entre el método Simplex basado en tableaus y el utilizado en este proyecto.

Las propiedades del método Simplex se pueden aplicar a problemas de PL, así sea $\mathcal{P}$ un problema de PL y B el conjunto de soluciones básicas del mismo, se puede particionar a B en dos conjuntos: B_f es el conjunto de soluciones básicas factibles y B_n el conjunto de soluciones básicas no factibles. Se define al pólitopo $\mathcal{P}$ como $\mathcal{P}_p$, es el conjunto de aristas cuyos vértices pertenecen a las soluciones básicas factibles. Todos los elementos de B están constituidos por una asignación de valores para cada una de las variables del problema. Suponiendo que existen n variables de decisión y el modelo de PL tiene m ecuaciones, al convertir las restricciones de desigualdad a igualdad, se agregan m variables de holgura. Al sumar las m variables de holgura a las n variables de decisión harán un total de $(m + n)$ incógnitas, por lo tanto, cada solución básica constará de $n + m$ variables i.e $b_i = (x_1, x_2, x_3, ..., x_n, s_1, s_2, s_3, ..., s_m)$ en las cuales cada variable tiene un valor definido. Para calcular una solución básica factible, resulta del total de $(m+n)$ variables, sólo n se igualan a cero $(n = 0)$, lo cual produce, un número finito de soluciones básicas con un límite máximo de $\binom{m+n}{m}$, como se denotó anteriormente en la Fórmula (1.2).

Las soluciones básicas pueden ser factibles y no factibles; se consideran sólo las primeras y se descartan las no factibles o segundas porque no están dentro de la región donde se cumplen todas las restrcciones del sistema. De la misma manera, a las variables que les fue asignado el valor de cero, las llamaremos no básicas y su valor ya es conocido. Se toman en cuenta sólo las soluciones básicas factibles, esto es, en las que tienen todas las variables básicas son mayor o igual 0; es decir, con un número de iteraciones menor a $\binom{m+n}{m}$ Fórmula (1.2), se obtienen soluciones básicas factibles: no degeneradas, si todas las incógnitas básicas son positivas y soluciones degeneradas, si al menos una variable básica es igual a 0, se aplican los criterios del algoritmo en forma iterativa para evaluar la función objetivo en puntos extremos adyacentes que potencialmente puedan mejorar el valor Z. Explicado lo anterior, para intercambiar una variable básica a una variable no básica implica que una variable no básica entre a la base y una variable de la base salga de la misma. Esto es, que una de las variables no básicas adquiere un valor diferente de cero y una de las variables básicas se hace 0.

Se generan nuevas soluciones básicas factibles, de tal manera que el valor de la función objetivo Z mejore; se repite el procedimiento (iteraciones), hasta que ninguna solu-

ción básica factible adyacente resulte mejor; es decir, hasta que no haya un incremento del valor.

considere el siguiente problema de la Tabla (3.5).

Tabla 3.5: Problema propuesto de programación lineal.

Normal	Normalizado
Maximize $z = 3x_1 + 2x_2$	Maximize $z = 3x_1 + 2x_2$
s.t. $5x_1 + 10x_2 \leq 30$	s.t. $5x_1 + 10x_2 + s_1 = 30$
$10x_1 + 5x_2 \leq 30$	$10x_1 + 5x_2 + s_2 = 30$
$x_1 + x_2 \leq 3.5$	$x_1 + x_2 + s_3 = 3.5$

En la Tabla (3.5) el tableau Normalizado está compuesto por columnas de variables de desición y de holgura, asi como de los coeficientes de la función objetivo Z. El problema de PL que se propone en la Tabla (3.5) en su forma de tableau se muestra en la Tabla (3.6).

Tabla 3.6: Problema de PL en su forma matricial de Tableau

	c_j	3	2	0	0	0	
c_B	x_B	x_1	x_2	s_1	s_2	s_3	RHS
0	s_1	5	10	1	0	0	30
0	s_2	10	5	0	1	0	30
0	s_3	1	1	0	0	1	3.5
	$c_j - z_j$	3	2	0	0	0	0

El método Simplex basado en tableaus y el método Simplex MSVNB, se basan en una serie de iteraciones para moverse entre soluciones básicas contiguas hasta que ya no es posible moverse. A cada una de estas soluciones básicas factibles le corresponderá un tableau distinto. Julian Hall *et al.* [Julian07] menciona que para cada iteración se tendrán que realizar los siguientes pasos como:

- Seleccionar la columna pivote c_N de una variable no básica $c_N \in \mathcal{N}$ que sea mayor a

cero.

- Encontrar la fila pivote c_B de la primer variable básica $c_B \in \mathcal{B}$ para que se convierta en cero.

- Intercambiar los índices c_N y c_B entre las matrices correspondientes a $A_{\mathcal{B}}$ y $A_{\mathcal{N}}$.

Algunas de las ventajas del método basado en tableaus son su fácil entendimiento y es simple de implementar. La desventaja que presenta es su alto costo, ya que la matriz A normalmente es calculada completamente. Los requisitos de almacenamiento son $O(m \times (m+n))$, y los requisitos computacionales son $O(m \times (m+n))$ operaciones de punto flotante por iteración. Éste método es numéricamente inestable si los vectores de las variables no básicas presentan dependencias lineales menciona Julian Hall en [Julian07]. Cada iteración del Simplex basado en tableaus, se hace a través de las operaciones elementales de renglones, esto es llevar a cabo un pivoteo parcial, es decir, se realizan las operaciones correspondientes para hacer una columna unitaria en la variable que entra, Harvey M. Wagner *et al.* [Wagner.56]. Después de aplicarla, la solución al sistema se puede encontrar en la columna del RHS.

3.2.2. Simplex Matricial

El método del Simplex matricial trabaja con la idea fundamental de que cualquier tabla del Simplex correspondiente a una solución factible básica, puede generarse de las ecuaciones originales por medio de operaciones matriciales, menciona Dantzig, G. B en [Dantzig53].

A diferencia del Simplex en su versión de tableaus, en la representación matricial sus cálculos son basados en la configuración inicial del tableau. Todas las columnas son transformaciones lineales de las columnas iniciales.

Hasta este punto, solo se ha explicadó el funcionamiento del Simplex basado en tableau; en el modelo del Simplex matricial, para actualizar el tableau la base será calculada mediante una serie de operaciones matriciales. El sistema de ecuaciones puede ser escrito como en la Fórmula (3.3).

$$A_{\mathcal{N}} x_N + A_{\mathcal{B}} x_B = b \tag{3.3}$$

Donde podemos expresar a x_B como en la Fórmula (3.4).

$$x_B = A_{\mathcal{B}}^{-1} b - A_{\mathcal{B}}^{-1} A_{\mathcal{N}} x_N \tag{3.4}$$

Haciendo $B = A_{\mathcal{B}}$, observamos que dicho cálculo involucra la matriz B^{-1}, con su repercusión en el costo computacional. Mas aún, se sabe de antemano que B^{-1} siempre se encuentra localizada en$A_{\mathcal{B}}$, específicamente en las columnas correspondientes a las variables básicas iniciales.

3.3. Simplex basado en columnas no básicas

En la versión del método Simplex MSVNB, el tableau únicamente se trabaja con las variables de las columnas no básicas y una columna auxiliar unitaria que llamaremos $\mathcal{K}$, por lo que el nuevo tableau queda de la siguiente manera:

Tabla 3.7: Tableau propuesto del método Simplex basado en columnas no básicas y una columna auxiliar.

	$\mathcal{N}$	$\mathcal{RHS}$	$\mathcal{K}$
1			$\vdots$
$\vdots$	$A_{\mathcal{N}}$	b	1
m			$\vdots$
0	c_N	$-f$	0

Lo anterior, marca una notable diferencia entre el Simplex basado en tableaus y el propuesto, ya que para cada iteración del Simplex no calcula todo el tableau, únicamente se ocuparán a las columnas no básicas, lo que implica una disminución en el número de operaciones y almacenamiento requerido.

Basándose en el problema propuesto de la Tabla (3.5) Normalizado, al comparar el tableau de la Tabla (3.6), se observa que es más grande que el tableau de la Tabla (3.8), ya que se eliminaron las variables básicas en las columnas.

Tabla 3.8: Problema de PL en la forma propuesta de Tableau y columna auxiliar unitaria.

	c_j	3	2		
c_B	x_B	x_1	x_2	RHS	$\mathcal{K}$
0	s_1	5	10	30	0
0	s_2	10	5	30	1s
0	s_3	1	1	3.5	0
	$c_j - z_j$	3	2	0	0

En la Tabla (3.6), se advierte que en cada una de las columnas base solamente se encuentra un uno, ya que, son columnas unitarias y por lo tanto, dicha variable básica solo existe en esa ecuación. Esto quiere decir que cada una de las variables básicas puede ser expresada de manera directa en términos de las variables no básicas. Aunque el tableau mostrado en la Tabla (3.8) únicamente utiliza columnas de las variables no básicas, no implica alguna alteración en las operaciones que normalmente se hacen en el Simplex basado en tableaus, sino que, ahora se llevarán a cabo menos operaciones. Entonces el tableau de la Tabla (3.8) se auxiliará de una columna unitaria $\mathcal{K}$, que figurará como una de las columnas básicas unitarias que se han eliminado, dicha columna llevará al uno en la posición calculada que corresponda al pivote en esa iteración i.e. la posición de la variable que deja la base.

El método Simplex basado en variables no básicas MSVNB (Non Basic Variable-based Simplex) se describe a continuación. Esta función recibe como parámetros de entrada una matriz A, el renglón r y la columna c que corresponden al pivote elegido para cada iteración. En las líneas 2 y 3 se obtiene la longitud del número de filas y columnas de la matriz A y se asignán en M y N, respectivamente. Posteriormente, en la línea 4 se le asigna a la columna auxiliar unitaria $\mathcal{K}$ el uno en la posición r, que corresponde a la variable básica que sale de la base. Luego, en las líneas 5 y 6 se efectúan los ciclos para recorrer todo el tableau y realizar las operaciones necesarias para hacer el pivoteo. Cabe señalar las operaciones que se desarrollan son exactamente las mismas que se utilizan en el método basado en tableaus. Asi mismo, se observa que estas operaciones involucran a la matriz A, con la ligera diferencia que también se realizan las operaciones en la columna unitaria $\mathcal{K}$.

Algoritmo 1: Método_Simplex_MSVN(A', c, b)

1: A $\leftarrow$ $\begin{array}{|c|c|}\hline A' & b \\ \hline c & 0 \\ \hline \end{array}$
2: **mientras** True **hacer**
3: $j \leftarrow argmax(c_j - z_j), c_j - z_j > 0, j \in \mathcal{N}$
4: **si** no definido(j) **entonces**
5: return x^*
6: **fin si**
7: $i \leftarrow argmin(\frac{b_i}{A_{ij}}), A_{ij} > 0, i = 1..m$
8: **si** no definido(i) **entonces**
9: return No hay solución
10: **fin si**
11: $A \leftarrow PivoteoParcial(A, i, j)$
12: **fin mientras**
13: **devolver** A

Una vez terminados los ciclos en las líneas 5 y 6, se obtiene una columna $\mathcal{K}$ que ya no es unitaria, ahora contiene valores distintos en cada renglón que la conforma. Por otro lado, la columna A_c se convierte en una columna unitaria, correspondiente a la columna de la variable no básica que sale de la base. Finalmente en la línea 12, una vez terminados los cálculos, se procede a asignar los valores de la columna $\mathcal{K}$ a la columna corresponiente a la variable que sale de la baseA_c.

La diferencia principal entre el algoritmo propuesto MSVNB y el del método matricial, consiste en que no se realiza el cálculo de la matriz inversa, ni guarda la configuración inicial, únicamente realiza un pivoteo parcial entre las columnas no básicas. Enseguida se expondrá un ejemplo del algoritmo propuesto, donde se asumirá la existencia de la solución inicial básica factible trivial.

Para la primera iteración se toman como variables básicas s_1, s_2 y s_3, a las no básicas x_1 y x_2 se da el valor cero, entonces se tiene la solución básica factible de partida $(x_1, x_2, s_1, s_2, s_3)$ con un valor para $z = x_1 + x_2 = 0$.

Algoritmo 2: PivoteoParcial (A, r, c)

```
1: M ← Filas(A)
2: N ← Columnas(A)
3: K = e_r
4: para j = 1...M hacer
5:   para i = 1...N hacer
6:     si i = r entonces
7:       A_ij ← A_ij / A_rc
8:       K_i ← K_i / A_rc
9:     si no
10:      A_ij ← A_ij − A_i,c * A_rj / A_rc
11:      K_i ← K_i − A_rj * K_i / A_rc
12:    fin si
13:  fin para
14: fin para
15: A_c ← K
16: devolver A
```

Posteriormente, para la segunda iteración se parte de la solución anterior, ya que es necesario seguir iterando para aproximarse a una mejor solución. Para esto se debe ingresar una nueva variable a la base, solo se pueden tener m variables, es necesario sacar otra variable de la base. Los cálculos para determinar el pivote y hacer las operaciones en el Simplex que se propone son exactamente los mismos, aplicándolos a un tableau más pequeño y haciendo las mismas operaciones en una columna auxiliar unitaria.

Los criterios para determinar la variables de entrada y salida son las mismas que en el método Simplex basado en tableaus, llamadas condiciones de optimalidad y de factibilidad, Taha.H *et al.* [Taha04] explica estos criterios a detalle. Este es un problema de maximización por lo que, la variable de entrada será la variable no básica que tenga el coeficiente $c_j - z_j$ mayor positivo y la variable de salida será la que tenga el coeficiente en la mínima razón $\frac{b_i}{A_{ij}}$ no negativa.

Tabla 3.9: T0 Tableau inicial y columna auxiliar unitaria

	c_j	3	2		
c_B	x_B	x_1	x_2	RHS	$\mathcal{K}$
0	s_1	5	10	30	0
0	s_2	**10**	5	30→	1
0	s_3	1	1	3.5	0
	$c_j - z_j$	↑3	2	0	0

En este caso, se ve que $c_1 - z_3$ es el que cumple tal condición y por lo tanto la variable asociada a tal columna es la que entra i.e. x_1. Por otro lado, para saber cuál es la variable que sale de la base, se debe ver cual restricción se violaría primero al moverse en la dirección de la variable que entra. Este criterio se evalúa al hacer la división término a término de la columna RHS entre la columna de la variable que entra, en este caso es x_1, y saldrá la variable relacionada con el renglón donde resulte la razón menor $\frac{b_i}{A_{ij}}$. Aplicando tal regla, entra x_1 y sale x_2, como lo señalan las flechas en la Tabla (3.9), ya que se ha encontrado cual variable entra y cual sale, sin embargo, ¿como reconfiguramos el tableau con esta información?. En el método basado en tableaus de la Tabla (3.6) se sabe que las columnas asociadas a las variables básicas son unitarias, con el uno en el renglón asociado a esa variable. Pero, en el tableau propuesto de la Tabla (3.8) se tomará la columna auxiliar unitaria con el uno en el renglón asociado a la variable básica de salida. Esto se ilustra en la Tabla (3.9). La solución se resuelve con un pivoteo parcial, aplicado al tableau propuesto, consiste en transformar las columnas de las variables no básicas y la columna auxiliar, este procedimiento transformará a la columna asociada a la variable en una columna unitaria, mientras que la columna auxiliar unitaria se transformará en nuevos valores como lo muestra en la Tabla (3.10). Se reforman todos los renglones incluyendo a la columna auxiliar de acuerdo al pivotes parcial, incluyendo el renglón $c_j - z_j$ obteniendo el tableau de la Tabla (3.10). Una vez que se terminan los cálculos se procede a asignar los datos de la columna auxiliar a la columna asociada a la variable básica. Cabe destacar que todos los valores $c_j - z_j$ asociados a las columnas de las variables básicas son ceros y se

obtienen del tableau final como la Tabla (3.11). Aplicando este procedimiento, se llega al tableau T1 y a la columna auxiliar K1 representados en la Tabla (3.11).

Tabla 3.10: T1 y K1 (primera iteración)

	c_j	3	2		
c_B	x_B	s_2	x_2	RHS	$\mathcal{K}$
0	s_1	0	7.5	15	-0.5
0	x_1	1	0.5	3	0.1
0	s_3	0	**0.5**	0.5 $\rightarrow$	-0.1
	$c_j - z_j$	0	$\uparrow$0.5	-9	-0.3

Tabla 3.11: Se copia el vector K a la columna de la variable no básica asociada.

	c_j	3	2		
c_B	x_B	s_2	x_2	RHS	$\mathcal{K}$
0	s_1	-0.5	7.5	15	0
0	x_1	0.1	0.5	3	0
0	s_3	-0.1	0.5	0.5 $\rightarrow$	1
	$c_j - z_j$	-0.3	$\uparrow$0.5	-9	0

Aplicando los criterios anteriormente mencionados, se concluye que entra x_2 y sale s_3, con lo cual el nuevo tableau será la Tabla (3.12) y finalmente se llega a la Tabla (3.13)

Al resultar negativas todas las entradas en el último renglón, ya no hay mayor ganancia en cambiar la base, pues se ha llegado a la solución óptima. Finalmente puede observarse en forma gráfica la ruta que siguió esta serie de pivoteos en la Figura (3.1).

3.4. Análisis y resultados

La solución práctica de los problemas reales que utilizan modelos de programación lineal, presentan una gran dificultad porque contienen mucha información que debe almacenarse en la computadora. Como ejemplo, se cita un problema con 10,000 variables

Tabla 3.12: T2 y K2 (segunda iteración)

	c_j	3	2		
c_B	x_B	s_2	s_3	RHS	$\mathcal{K}$
0	s_1	1	0	-15	-15
0	x_1	0.2	0	-1	-1
0	x_2	-0.2	1	2	2
	$c_j - z_j$	-0.2	0	-9.5	-1

Tabla 3.13: Se copia el vector K a la columna de la variable no básica asociada.

	c_j	3	2		
c_B	x_B	s_2	s_3	RHS	$\mathcal{K}$
0	s_1	1	-15	-15	0
0	x_1	0.2	-1	-1	0
0	x_2	-0.2	2	2	0
	$c_j - z_j$	-0.2	-1	-9.5	0

de decisión y 500 restricciones, aquí la matriz A tendrá 5 millones de elementos, quizá muchas de ellos cero. Éstos tendrían que ser almacenados en la memoria con sus conocidas consecuencias. Como se observó, nuestra propuesta trabaja solamente con la parte del tableau asociado a las variables no básicas, lo cual le otorga ventajas, tal como la solución de problemas lineales grandes con el consecuente ahorro de cálculos. En contrario, el método matricial ocupa memoria para almacenar B, en cada iteración se tiene que hacer el cálculo de B^{-1} donde B es la base correspondiente a esa iteración. En este método B^{-1} es indispensable, pues conociendola tanto z como $z_j - c_j$ pueden ser calculados, más información en Witenberg, J. P *et al.* [Witenberg00]. En lo que concierne al método Simplex basado en tableaus, almacena en cada iteración un tableau de $m \times (m + n)$. En el método propuesto, la memoria ocupada es de $m \times (n + 1)$, es decir, la modificación que se plantea en este trabajo repercute en un menor número de operaciones que se realizan, ya que se propone un tableau más pequeño, basado únicamente en las variables no básicas y a diferencia del

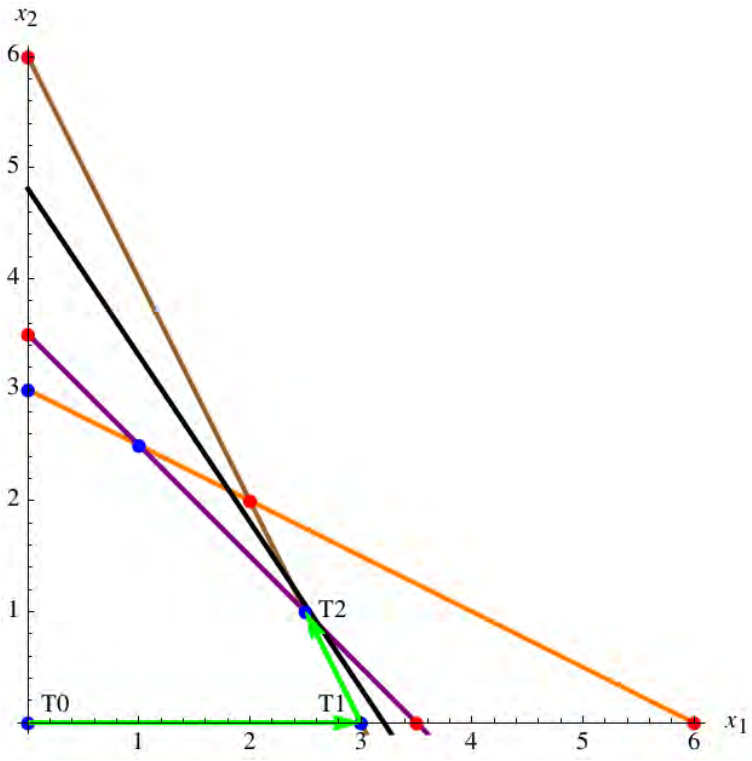


Figura 3.1: Ruta seguida por el método Simplex MSVNB .

Simplex matricial, no se calcula una matriz inversa.

3.4.1. Resultados preliminares a la modificación del método Simplex

En esta sección se ha propuesto una versión modificada del método Simplex basado en tableaus, para su implementación, solamente se utilizan las columnas no básicas del mismo, así como una columna auxiliar. Como resultado de la modificación hay un ahorro de memoria comparado con el Simplex basado en el tableau, por tanto, se realizarán menos operaciones. Lo anterior es así, debido a que solo se lleva a cabo el pivoteo parcial en cada una de las iteraciones, si se compara con el método Simplex matricial. Es importante remarcar que no se está comparando la propuesta con otros enfoques para resolver problemas de PL basados en punto interior, sino que, esta propuesta, a pesar de que no se trabaja con el tableau completo, no afecta ni el comportamiento del método Simplex ni los resultados, al contrario, ahora se tiene la ventaja de realizar menos operaciones y almacenar menos

información entre cada iteración; en consecuencia, se puede afirmar que la eficiencia del método Simplex basada en tableaus es mejorada.

Esta técnica puede utilizarse para problemas de n variables, esto significa que se puede generalizar para trabajar con cualquier número de dimensiones que sean necesarias.

3.5. Comentarios finales

En este capítulo se presentó una descripción del esquema propuesto para trabajar con el método Simplex basado en columnas no básicas (MSVNB). Se inició con la definición de algunos términos comunes y posteriormente, se expuso a detalle cada uno de los pasos propuestos en conjunto, con sus respectivos algoritmos y ejemplos ilustrativos de funcionamiento. Este método es un buen punto de partida ya que reduce el número de operaciones necesarias para encontrar el primer vértice de intersección ahorrando espacio de memoria y tiempo de procesamiento, a diferencia de otras variantes del Simplex. Habiendo discutido los detalles de diseño del método Simplex (MSVNB), en el siguiente capítulo se presenta el desarrollo de la modificación a este método y como se realiza el proceso de búsqueda para encontrar los vértices de intersección entre la hipercaja y el hiperplano.

Capítulo 4

Cálculo de la intersección entre una hipercaja y un hiperplano

Para abordar el tema a analizar, se inicia con una breve explicación en convertir el problema de la intersección entre una hipercaja y un hiperplano en un problema de PL. Después se realiza el proceso para determinar las intersecciones entre la hipercaja y el hiperplano a través de la modificación al método Simplex (MSVNB) propuesta en este proyecto. También se explica cómo se lleva a cabo el recorrido entre todos los vértices de intersección de acuerdo a las técnicas basadas en DFS y finalmente se describen las estructuras de control que se utilizarán sobre los vértices de intersección que se vayan recorriendo.

4.1. Convertir el problema de la intersección en un problema de PL.

Para resolver el problema de la intersección entre una hipercaja y un hiperplano desde un enfoque de PL, se propone modelar el problema de la intersección en una serie de ecuaciones lineales, lo cual permitirá describir una hipercaja y un hiperplano mediante una serie de desigualdades lineales.

Al tomar como base el problema de PL en su forma estándar, definido en la

Fórmula (3.1), se hace la propuesta presentada para convertir el problema de intersección entre una hipercaja y un hiperplano, donde las ecuaciones lineales que describirán a la función objetivo Z estará dada por el hiperplano h, mientras que las ecuaciones lineales que describirán a las restricciones del problema estarán dadas por las características de la hipercaja y por el hiperplano h.

Por ejemplo: se tiene el siguiente sistema de ecuaciones lineales como se presenta en el Ejemplo 4.1, el cual describe un sistema que genera la intersección entre la hipercaja e hiperplano presentado en la Figura (4.1). Para efecto de este ejemplo,

$$\begin{aligned} \underset{x}{\text{Maximizar}} \quad & Z = 2x_1 + 2x_2 + 2x_3 \\ \text{Sujeto a} \quad & 2x_1 + 2x_2 + 2x_3 < 3 \\ & x_1 \leq 1 \\ & x_2 \leq 1 \\ & x_3 \leq 1 \end{aligned} \tag{4.1}$$

Por lo que se pretende tener un sistema de ecuaciones donde tanto la función objetivo Z como una de las restricciones definidas en el problema estarán asociadas a la misma ecuación descrita para el hiperplano h.

El propósito de hacer que la función objetivo sea el hiperplano h y que a la vez sea una de las restricciones, es con el fin de que al realizar la maximización a través del método Simplex (MSVNB) sobre el sistema, el proceso comience a maximizar la función objetivo y en consecuencia la solución que arroja este sobre la restricción asociada al hiperplano. Pues la función objetivo y la restricción asociada al hiperplano son paralelas y el resultado será un punto que esté sobre el mismo hiperplano, ese punto será el primer vértice de intersección entre la hipercaja y el hiperplano.

Como el hiperplano es una de las ecuaciones lineales del sistema (restricción), limita el área de búsqueda a los vértices de intersección que caen dentro del hiperplano, el cual es parte de la región factible. Al hacer la función objetivo Z igual al hiperplano H se pretende que la primera solución que arroje método Simplex (MSVNB), llegue a un punto sobre el mismo hiperplano. De manera que se asegura de dar como primer resultado un

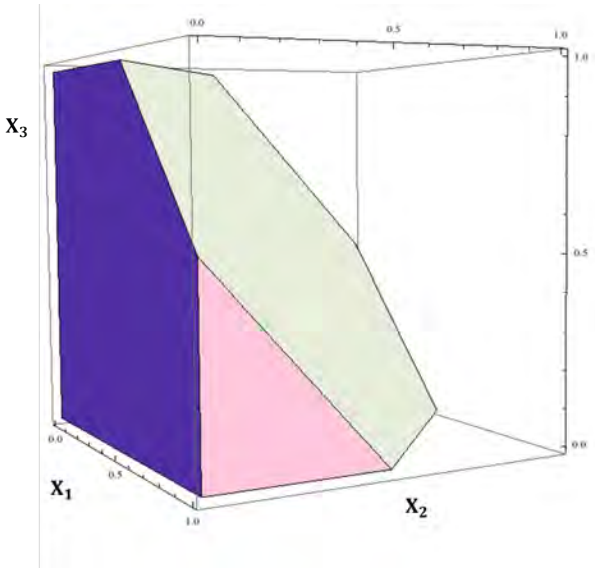


Figura 4.1: Intersección entre una hipercaja y un hiperplano.

vértice de intersección entre la hipercaja e hiperplano.

Gráficamente en la Figura (4.1) se tiene un hiperplano que corta la hipercaja produciendo seis vértices de intersección, además, dadas las restricciones del problema, se sabe que la región factible es toda el área ubicada por debajo del hiperplano.

Entonces, primero es necesario llegar a un vértice de las esquinas de intersección entre la hipercaja e hiperplano y, a partir de esa solución encontrar los cinco vértices restantes para este caso en específico.

En la Figura (4.2) se observa el recorrido que realiza el método Simplex (MSVNB) presentado en el Capítulo 3, para llegar a la primer esquina perteneciente al politopo.

A partir de que se encuentre el primer vértice de intersección, se aplica la modificación al método Simplex (MSVNB) replanteando algunos de los criterios por los que se rige dicho método, para que a partir de ese punto únicamente se procesen todos los vértices de intersección que existan entre la hipercaja e hiperplano.

En la siguiente sección se explicará cómo se lleva a cabo el análisis en el sistema de ecuaciones, para realizar la modificación al método Simplex (MSVNB) e identificar únicamente los vértices de intersección que son de interés para este proyecto.

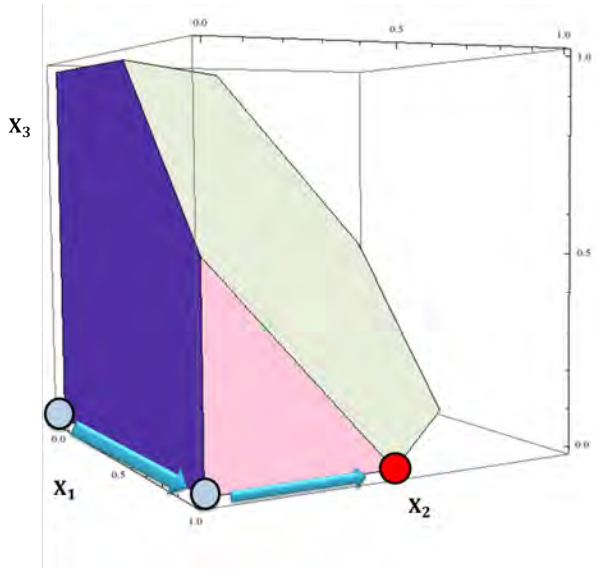


Figura 4.2: Recorrido efectuado por el método Simplex basado en columnas no básicas MSVNB, entre los vértices de soluciones básicas hasta llegar a un vértice de intersección.

4.2. Método Simplex (MSVNB) modificado para determinar las intersecciones entre la hipercaja e hiperplano

En este apartado se analizara la modificación al método Simplex (MSVNB) con la finalidad de poder determinar cuáles son los vértices de intersección adyacentes que conforman el politopo de la intersección entre una hipercaja y un hiperplano.

La modificación al método Simplex (MSVNB) consiste en cambiar las reglas tradicionales por las que se rige dicho método, de tal forma que se dé un enfoque distinto a dichas reglas y a la interpretación de los datos que entrega adecuándolo a las necesidades particulares del problema de la intersección.

A continuación se enlistan las cuatro principales propiedades por las que se rige el método Simplex (MSVNB), Vasek Chvatal en [Chvatal83] :

1. Criterio de factibilidad. No debe evaluar soluciones que no sean factibles.

2. Criterio de ganancia. Conforme se mueve a través del politope debe siempre mejorar la evaluacion de Z.

3. Criterio de optimalidad. Siempre se desplaza a soluciones básicas factibles vecinas.

4. Criterio de paro. Debe ser capaz de identificar cuando ha alcanzado el óptimo y terminar.

La modificación consistirá en replantear estos cuatro principales criterios del Simplex, pues una vez que se tiene un primer vértice de intersección sobre el hiperplano entregado por el método Simplex (SMVNB) lo que se requiere es cambiar las reglas del Simplex para que la b´usqueda de los vértices de intersección restantes se mantenga sobre el área del hiperplano, ya que si se siguiera aplicando las reglas normales de Simplex (MSVNB) este se saldría del área de búsqueda del hiperplano al tratar de seguir el proceso normal de la maximización.

Por lo que se replantean los principales criterios de la siguiente manera:

1. Criterio de factibilidad. No debe evaluar vértices que no sean factibles ni tampoco se debe evaluar la variable que estén asociada al hiperplano. Esto quiere decir que no se deben ingresar variables a la base que no sean factibles y además no se debe ingresar la variable asociada a la restricción del hiperplano ya que de otra manera el vértice se movería fuera del hiperplano.

2. Criterio de ganancia. Conforme se mueve a través del politope ya no se genera perdida ni ganancia en la evaluación de la función objetivo Z, pues se pretende estar sobre la misma área de búsqueda que será el hiperplano y no salir de este.

3. Criterio de optimalidad. Siempre se desplaza a soluciones básicas factibles vecinas siempre cuando, la nueva variable que se desee entrar a la base tengan un valor de 0 en el reglón $c_j - z_j$, lo que querrá decir que al entrar a la base este nuevo vértice se mantendrá sobre el área del hiperplano y no generara ninguna ganancia en Z.

4. Criterio de paro. Debe ser capaz de identificar cuando ha alcanzado todos los vértices de intersección utilizando técnicas de recorrido de grafos basadas en DFS y tablas hash para el control de los vértices y de esta manera terminar.

Como ya se mencionó antes, a partir del primer vértice de intersección solución encontrado por el método Simplex (MSVNB), el siguiente paso en el es determinar todas las soluciones de los vértices de intersección adyacentes que existan en el sistema. Para

realizar esta tarea es necesario analizar el tableau que arroja el método Simplex (MSVNB) correspondiente al primer vértice de intersección analizando los resultados en la fila $c_j - z_j$.

La finalidad de modificar los criterio se aplican en el método Simplex, primero es determinar las variables que entrarán y saldrán de la base entre cada iteración para no visitar vértices que están fuera del hiperplano con el fin de no generar una ganancia ni perdida en la función Z y con esto asegurar que se está sobre el hiperplano únicamente visitando vértices adyacentes, además se utiliza un algoritmo de recorrido de grafos para determinar cuándo se han encontrado todos los vértices de intersección, este algoritmo se explica en la sección 4.3.

La modificación se ve reflejada en las operaciones que se aplican en el Tableau, que consiste en elegir entre las columnas de las variables no básicas la variable de entrada donde $c_j - z_j = 0$. La variable no básica asociada a un valor 0 en el renglón $c_j - z_j$ quiere decir que, si se ingresa a la base no se obtendrán ganancias ni pérdidas, ya que no tiene un valor que haga a la función objetivo Z se desplace creciendo o disminuyendo su valor, por lo tanto se mantendrá sobre el hiperplano, y se estará viajando únicamente entre los vértices de intersección factibles adyacentes y que además se encuentran sobre el mismo hiperplano.

Este concepto quiere decir que si el algoritmo ingresa una variable no básica con valor de 0 en $c_j - z_j$ la solución siguiente que se genere en el tableau estará sobre el mismo hiperplano, y será un punto de intersección entre el hiperplano y la hipercaja. Por lo que cada movimiento entre los vértices representa al mismo sistema que se pretende resolver pero desde distintos puntos de vista.

A continuación en este apartado se analizan los tableaus que se toman como soluciones, para verificar que se está sobre los vértices de intersección entre la hipercaja e hiperplano. Al poner el problema de PL del Ejemplo 4.1 en su forma de tableau, se obtiene la Tabla (4.1). Una vez que el sistema de la Tabla (4.1) es resuelto a través del método Simplex (MSVNB), arroja como resultado la solución del sistema, mostrada en la Tabla (4.2). En la Figura (4.1) se puede observar el recorrido que realizó entre los vértices para llegar al resultado final, donde en cada iteración se selecciona una variable no básica del problema que produce un mayor progreso hacia el vértice de intersección entre la hipercaja e hiperplano.

Tabla 4.1: Representación de Tableau del problema de PL entre una hipercaja y un hiperplano

	c_j	2	2	2		
c_B	x_B	x_1	x_2	x_3	RHS	$\mathcal{K}$
0	s_0	2	2	2	3	0
0	s_1	1	0	0	1	0
0	s_2	0	1	0	1	0
0	s_3	0	0	1	1	0
	$c_j - z_j$	2	2	2	0	0

Esta solución es considerada una SBF óptima conformada de una configuración de variables básicas y no básicas que, además de ser el primer vértice de intersección entre la hipercaja e hiperplano encontrado, se ubica dentro de la región factible.

Tabla 4.2: Primera configuración en forma de tableau del vértice de intersección entre una hipercaja e hiperplano, encontrado por el método Simplex (MSVNB).

	c_j	2	2	2		
c_B	x_B	s_1	s_0	x_3	RHS	$\mathcal{K}$
0	x_2	-1	0.5	1	0.5	0
0	x_1	1	0	0	1	0
0	s_2	1	-0.5	-1	0.5	0
0	s_3	0	0	1	1	0
	$c_j - z_j$	0	-1	0	-3	0

Por ejemplo en la Tabla (4.2) se puede observar que el vértice en el que se encuentra el sistema contiene información de dos vértices adyacentes, que son también vértices de intersección. Esto se puede apreciar si analizamos el renglón $c_j - z_j$ ya que existen dos columnas en las variables no básicas s_1 y x_3 que tienen un valor de 0.

Entonces los dos vértices de intersección están relacionados a las variables no básicas s_1 y x_3 respectivamente, cualquiera de las dos posibilidades se tendrá que computar

y el orden dependerá de las técnicas de recorridos de grafos que se implementen. En la sección 4.3 se explicará a fondo cuales son estas técnicas y cuál es la que se implementó en este proyecto.

Gráficamente se puede observar en la Figura (4.3) a los dos vértices de intersección vecinos que se tienen y deben computar. Para el ejemplo se opta por computar las variables de izquierda a derecha. Entonces dado el tableau de la Tabla (4.2) primero se dejará entrar a la base a s_1 y después a x_3.

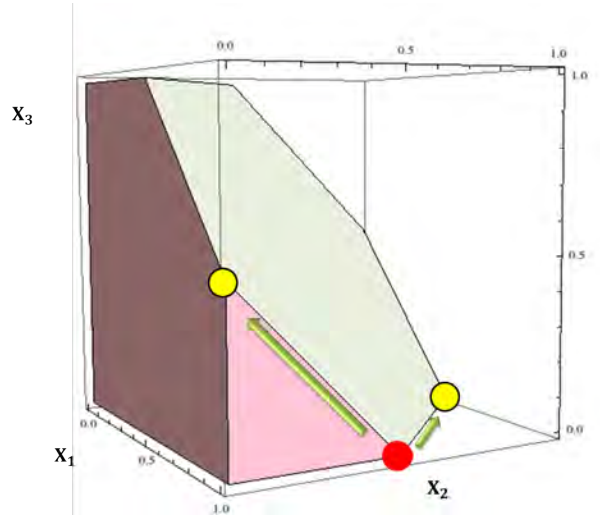


Figura 4.3: Vértices de intersección adyacentes.

Primero se computa a s_1, querrá decir que la variable que ahora entrará a la base sera s_1 y la variable de salida será s_2.

Si se computa el tableau presentado en la Tabla (4.2), dejando entrar a la base a s_1 y sacando de la base a s_2 lo que se obtendrá es un nuevo vértice de intersección entre la hipercaja e hiperplano con su respectiva configuración en el tableau como se muestra en la Tabla (4.3). Gráficamente se podría ver como que se traslada de un vértice de intersección a otro en la Figura (4.4).

Del tableau presentado en la Tabla (4.3) se puede observar que el resultado que se obtiene de la función objetivo Z es el mismo, no se obtiene una mejora ni empeora. Con esto se quiere decir que gráficamente el vértice encontrado no se encuentra por debajo del hiperplano o arriba de éste sino que se mantiene sobre él. Gráficamente en la Figura (4.4) se

Tabla 4.3: Tableau de vértice adyacente s_1 a la base y extrayendo a s_2.

	c_j	2	2	2		
c_B	x_B	s_2	s_0	x_3	RHS	$\mathcal{K}$
0	x_2	1	0	0	1	0
0	x_1	-1	0.5	1	0.5	0
0	s_1	1	-0.5	-1	0.5	0
0	s_3	0	0	1	1	0
	$c_j - z_j$	0	-1	0	-3	0

observa como el algoritmo se traslada hacia el otro vértice de intersección vecino al original.

Ahora se puede seguir haciendo el análisis para verificar si este nuevo vértice contiene vértices adyacentes que puedan ser intersecciones válidas. Si se analiza el tableau de la Tabla (4.3), se puede observar que tiene otros dos vecinos de intersección, esto quiere decir que habrá que computar también a estos dos nuevos vértices de intersección y así sucesivamente hasta que todos los vértices de intersección estén computados.

Sin embargo, si computáramos los dos nuevos vecinos que aparecen se vería que uno de los dos vecinos ya fue computado y corresponde al tableau de la Tabla (4.2) que es el vecino de intersección anterior, con el que se partió, y si se llegara a computarlo esto haría que se ciclara infinitamente donde el algoritmo se mantendría viajando entre dos vértices de intersección adyacentes.

Lo mismo sucedería si se comenzara a computar la variable no básica relacionada a x_3 del tableau presentado en la Tabla (4.2), se obtendría una configuración de tableau distinta, correspondiente a esta nueva base se generara el tableau mostrado en la Tabla (4.4).

Gráficamente en la Figura (4.5) se observa como el algoritmo se trasladaría hacia el otro vértice de intersección vecino al original.

Esta nueva configuración en el tableau de la Tabla (4.4) corresponde a un nuevo vértice de intersección que contiene otros dos vecinos que habrá que computar también, sin embargo, se aprecia que igualmente uno de ellos, al igual que en tableau de la Tabla (4.3),

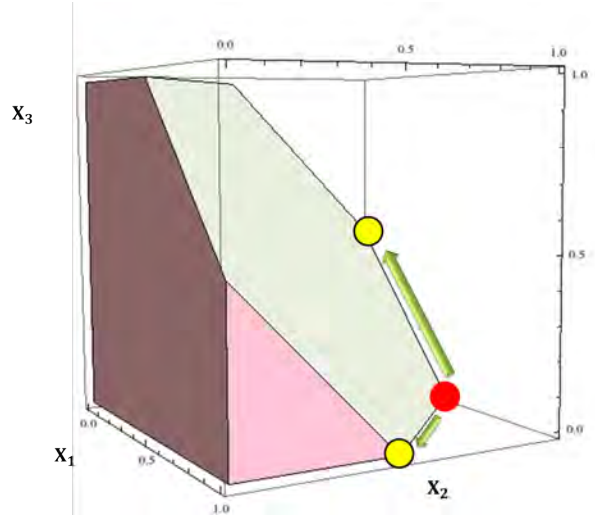


Figura 4.4: Vértices de intersección adyacentes, asociados a la variable de entrada s_1.

corresponde al vértice de intersección anterior y si se vuelve a computar esa configuración de igual forma se ciclara infinitamente.

Por lo tanto para hacer un análisis correcto entre cada vértice de intersección y sus vecinos correspondientes se debe contar con una técnica de recorrido de vértices y tablas de control de los mismos, con la finalidad de evitar ciclos. A continuación se explican las técnicas utilizadas en este proyecto para la realización del recorrido entre los vértices y cuáles fueron las técnicas para el control de los mismos.

4.3. Recorrido de vértices.

Para evitar que los vértices de intersección entre la hipercaja e hiperplano no sean procesados más de una vez es necesario utilizar alguna de las técnicas de recorrido de grafos ya existentes, donde se pueda visitar todos los vértices que cumplan con ciertas características y sean visitados una sola vez.

El hecho de que todos los vértices deban ser computados una sola vez, y no deban repetirse es para evitar caer en ciclos infinitos. Pero para hacer este recorrido entre las soluciones se debe de buscar alguna técnica que ayude a recorrer todos los vértices de intersección entre la hipercaja e hiperplano.

Tabla 4.4: Tableau de vértice adyacente x_3 a la base y extrayendo a x_2

	c_j	2	2	2		
c_B	x_B	s_1	s_0	x_2	RHS	$\mathcal{K}$
0	x_3	-1	0.5	1	0.5	0
0	x_1	1	0	0	1	0
0	s_2	0	0	1	1	0
0	s_3	1	-0.5	-1	0.5	0
	$c_j - z_j$	0	-1	0	-3	0

Una vez que se encuentre una de las SBF óptimas, es decir un vértice de intersección, para hacer el recorrido de todos los vértices de intersección dentro del politopo y procesar a cada uno de ellos existen técnicas de recorridos de grafos que pueden utilizarse, Carlos García *et al.* [García02] menciona algunos ejemplos como la búsqueda en anchura BFS (Breadth First Search), la búsqueda en profundidad DFS (Depth First Search), técnicas como las utilizadas en teoría de grafos para encontrar un camino euleriano, un camino hamiltoniano, árbol de expansión también llamado árbol de recubrimiento (Spanning tree).

Ernesto Coto *et al.* [Coto03] menciona que en ciencias de la computación, el recorrido de árboles se refiere al proceso de visitar de una manera sistemática, exactamente una vez, cada nodo en una estructura de datos de árbol examinando y/o actualizando los datos en los nodos. Tales recorridos están clasificados por el orden en el cual son visitados los nodos.

Sin embargo entre todas estas técnicas que existen la que mejor se adecua a el problema de este proyecto es la utilizada en los árboles de expansión y búsqueda a profundidad ya que requieren menos memoria de procesamiento en comparación con la búsqueda a lo ancho.

A continuación se describen las propiedades y el funcionamiento de la búsqueda a profundidad y como se hace la implementación para este proyecto.

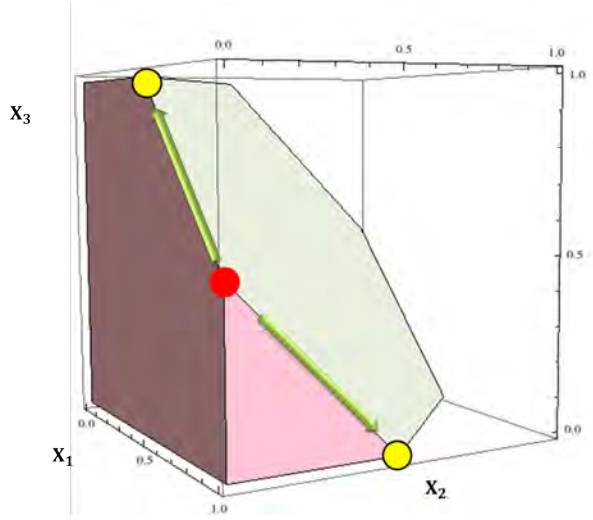


Figura 4.5: Vértices de intersección adyacentes, asociados a la variable de entrada x_3.

4.3.1. Búsqueda en profundidad.

Como se mencionó anteriormente, una vez que se obtiene el primer vértice solución del sistema de ecuaciones, ahora el proyecto se enfocará en recorrer todos los vértices de intersección restantes que son soluciones que existan en el sistema, ya que puede haber más de una solución dependiendo de la dimensión con la que se esté experimentando.

En este proyecto se implementó una técnica para recorrer los vértices de intersección, basada en el recorrido a profundidad (DFS). A continuación se explica como se lleva a cabo la implementación de esta técnica para que se pueda hacer el recorrido entre los vértices.

Ralph Grimaldi *et al.* [Grimaldi98] menciona que el recorrido de los árboles basado en DFS es un algoritmo que permite recorrer todos los nodos o vértices de un grafo o árbol de manera ordenada, pero no uniforme. Su funcionamiento consiste en ir expandiendo todos y cada uno de los nodos que va localizando, de forma recurrente, en un camino concreto. Cuando ya no quedan más nodos que visitar en dicho camino, regresa (Backtracking), de modo que repite el mismo proceso con cada uno de los vecinos del nodo ya procesado.

Esta técnica es implementada en el proceso para recorrer todos los vértices de intersección entre la hipercaja e hiperplano, ya que este proceso hace el recorrido entre

todos los vértices visitándolos una sola vez sin hacer ciclos en los cuales podamos caer en errores.

El algoritmo DFS posee varias aplicaciones las más importantes son: para problemas de conectividad, si un grafo es conexo, detectar ciclos en un grafo, número de componentes conexas, etc. También es bastante útil en otro tipo de algoritmos para encontrar las componentes fuertemente conexas en un grafo (Algoritmo de Kosaraju, Algoritmo de Tarjan), para hallar puntos de articulación o componentes biconexas (puentes), para recorrido en un circuito o camino euleriano, topological sort, flood fill y otras aplicaciones.

Algunas de las aplicaciones en la vida cotidiana se enfocan generalmente en problemas de optimización como por ejemplo en las redes de comunicación eléctrica, telefónica, carretera, ferroviaria, aérea, marítima, etc.; donde los nodos representan puntos de consumo eléctrico, teléfonos, aeropuertos, computadoras. Otro ejemplo es en la construcción de carreteras pavimentadas que unen varias poblaciones. El camino entre dos poblaciones puede pasar por una o más poblaciones adicionales.

DFS va formando un árbol al igual que BFS pero lo hace a profundidad. Existen dos formas de hacer el recorrido una es usando una pila de forma iterativa y la otra es de manera recursiva. En el Algoritmo (3) DFS comienza al visitar el nodo inicial y lo ingresa en la pila, para ver los siguientes nodos a visitar se extrae el nodo tope de la pila y se ven sus adyacentes, este tipo de pila revisa los nodos adyacentes y los que no han sido visitados los inserta en la pila. El proceso se repite hasta que la pila se encuentre vacía (se han visitado todos los nodos).

La pila que se utiliza es una estructura de datos en la que el modo de acceso a sus elementos es de tipo LIFO (del inglés Last In First Out, último en entrar primero en salir) que permite almacenar y recuperar datos. Niklaus Wirth *et al.* [Wirth80] brinda mayor información acerca de las estructuras de datos.

Para realizar un recorrido entre los vértices de intersección exitoso, es necesario tener tablas de control de los mismos con la finalidad de tener la información actualizada de todos los vértices y de esta manera tener conocimiento de las propiedades del sistema con el que se está tratando. En la siguiente sección se explicará a detalle el uso de las tablas de control que se implementaron en este proyecto.

Algoritmo 3: Algoritmo para el recorrido de vértices basado en DFS.

```
1: creamos una pila S
2: agregamos origen a la pila S
3: marcamos origen como visitado
4: mientras :
5: mientras S no este vacio hacer
6:    sacamos un elemento de la pila S llamado v
7:    para para cada vértice w adyacente a v en el Grafo hacer
8:       si si w no ah sido visitado entonces
9:          marcamos como visitado w
10:         insertamos w dentro de la pila S
11:      fin si
12:   fin para
13: fin mientras
```

4.4. Implementación de Control.

Roberto Hernández *et al.* [Hernández01] menciona que una tabla hash, es una estructura de datos que se puede ver como un contenedor (Diccionario) asociativo, que permite un almacenamiento y posterior recuperación eficiente de elementos a partir de otros objetos, llamados claves.

La búsqueda es la operación principal que soporta de manera eficiente, ya que permite el acceso a los elementos almacenados a partir de una clave generada. Funciona transformando la clave con una función hash en un indice, un número que identifica la posición (casilla o cubeta) donde la tabla hash localiza el valor deseado.

Una tabla hash se puede ver como un conjunto de entradas. Cada una de estas entradas tiene asociada una clave única, y por lo tanto, diferentes entradas de una misma tabla tendrán diferentes claves. Esto implica, que una clave identifica unívocamente a una entrada en una tabla hash. Las entradas de las tablas hash están compuestas por dos componentes, la propia clave y la información que se almacena en dicha entrada.

Cabe destacar que las tablas hash proveen tiempo constante de búsqueda promedio $O(1)$, sin importar el número de elementos en la tabla. Comparada con otras estructuras de listas enlazadas, las tablas hash son más útiles cuando se almacenan grandes cantidades de información. Otras estructuras como árboles binarios auto-balancéales tienen un tiempo promedio de búsqueda mayor.

La estructura de las tablas hash es lo que les confiere su gran potencial, ya que hace de ellas unas estructuras extremadamente eficientes a la hora de recuperar información almacenada. El tiempo medio de recuperación de información es constante, es decir, no depende del tamaño de la tabla ni del número de elementos almacenados en la misma.

Por lo tanto, las tablas hash son muy útiles cuando el tiempo de acceso a la información es crítico. La gran eficiencia que proporcionan estas tablas hacen que sean las estructuras de datos escogidas en situaciones tales como la implementación de la tabla de símbolos de un compilador. Esta es una tarea para la cual se adaptan a la perfección gracias a su carácter asociativo y su eficiencia.

Con el objetivo de llevar el control de los vértices de intersección que se van computando se implementaron 3 tablas hash, en donde a todos los vértices de intersección se les asigna una clave única de tal manera que se facilite el acceso para obtener información específica en cada uno de ellos.

Se utilizó una tabla hash a la cual se le ingresa una clave única por cada vértice, esta clave estará compuesta de las variables básicas y no básicas que presenta cada vértice en su configuración. Se asigna un valor de 0 a las variables no básicas y un valor de 1 a las variables que pertenecen a la base, de esta manera se pretende formar una clave binaria compuesta de ceros y unos, la cual será única por cada elemento, con esto se pretende evitar ingresar a la tabla vértices repetidos.

Adicionalmente se desarrolló una segunda tabla hash que tenga el control de los vértices visitados y no visitados, de esta manera se evitará visitar más de una vez el mismo vértice y hacer operaciones innecesarias.

Y finalmente se realizó tener una tercera tabla hash la cual lleva el control de la configuración numérica por cada vértice de intersección, esta configuración contendrá los valores de las variables que lo conforman, de esta manera se pretende tener todos los valores

de las variables asociadas a cada vértice y por lo tanto en esta tabla se tendrá las soluciones al sistema.

Todas estas tablas estarán asociadas a una clave única, de esta manera se obtendrá la información actual de cada vértice que conforme la intersección entre la hipercaja e hiperplano.

Tener el control en cada uno de los vértices y su información actual es de suma importancia para saber en qué estado se encuentra el sistema que se está tratando de resolver. A continuación se explica a detalle cada una de las tablas hash que se proponen.

4.4.1. Tabla de configuración binaria

La tabla hash propuesta para el control de una clave única por cada elemento, obtiene la configuración binaria de cada vértice de intersección que se ingresa a la pila de almacenamiento. Otorgando a cada vértice ingresado un ID para llevar el control del número de vértices ingresados y una clave única para identificar a cada vértice. La clave única estará dada por la configuración de las variables que se encuentran en la base y las que están fuera de ella.

Esta clave única que se propone estará compuesta de ceros y unos, donde se pretende dejar a las variables básicas con un 1 y las variables no básicas con un 0.

Por ejemplo basados en el ejemplo de la Tabla (4.2), presentado en el Capítulo 4, la configuración de entrada de un vértice contiene en total 7 variables básicas y no básicas, de las cuales 4 pertenecen a la base y 3 están fuera de ella, estas se enumerarán en orden como se muestra a continuación en la Tabla (4.5), en la cual se forma una clave binaria dada una configuración de variables básicas y no básicas.

Entonces esta clave binaria se ingresa a la tabla hash y se le asocia con un valor (ID) para cada vértice, como se ve en la Tabla (4.6).

Con esto se garantiza tener el control de todos los vértices que se ingresen, teniendo una clave binaria única evitando ingresar vértices repetidos.

Tabla 4.5: Ejemplo de formación de clave única

Variables		
variables básicas:	x_2, x_1, s_2, s_3	
variables no básicas:	s_1, s_0, x_3	
		Clave unica
Variables ordenadas:	$x_1, x_2, x_3, s_0, s_1, s_2, s_3$	$\rightarrow$ 1100011

Tabla 4.6: Tabla de control para la configuración binaria

Clave binaria		Valor (ID)
1100011	$\rightarrow$	1
$\vdots$		$\vdots$

4.4.2. Tabla de visitas

La tabla hash propuesta para el control de los vértices de intersección visitados y no visitados, se realiza cuando un vértice se ingresa y se extrae de la pila de almacenamiento.

Cuando el vértice de intersección es ingresado a la pila se le asigna un valor de 0 significa que el vértice acaba de ser ingresado a la pila y aún no ha sido visitado. Así mismo cuando el vértice es extraído de la pila se le asigna un valor de 1 lo que significa que el vértice ha sido extraído de la pila y ha sido visitado para explorar sus vértices de intersección adyacentes.

Este control ayuda a obtener la información del estado en el que se encuentran los vértices. Por ejemplo, si un vértice ha sido visitado o no, en esta tabla se podrá consultar su estado, de esta manera se evita visitar a un vértice de intersección más de una vez.

La tabla hash implementada para llevar el control de los vértices visitados se puede observar en el ejemplo de la Tabla (4.7), la cual consta de una clave que será el mismo ID relacionado a la configuración binaria en Tabla (4.6) y este se asociará a un valor el cual será un 0 si el vértice de intersección aún no ha sido visitado y 1 si el vértice de intersección ya fue visitado.

Tabla 4.7: Tabla de visitas

Clave binaria		Clave (ID)		Visitados
0100010101	→	1	→	0
0100010101	→	2	→	0
⋮		⋮	→	⋮

De esta manera se logra el objetivo para evitar visitar vértices de intersección que ya fueron visitados, evitando hacer operaciones innecesarias en la busqueda de vecinos adyacentes asi como ciclos infinitos.

4.4.3. Tabla de configuración numérica

La tabla hash propuesta para el control de los valores numéricos de las variables del sistema, obtiene la configuración numérica de cada vértice que se ingrese a la pila de almacenamiento.

Al ingresar un vértice en la pila se le asigna un ID, el cual está asociado a la clave única en la Tabla (4.6) y una clave numérica para identificar las variables cada vértice. Esta clave numérica se asigna de acuerdo a los valores que presenten las variables básicas y no básicas.

Basados en el ejemplo de la Tabla (4.2) del Capítulo 4, la configuración de entrada de un vértice contiene en total 7 variables básicas y no básicas, de las cuales 4 pertenecen a la base y 3 están fuera de la base, estas se enumerarán en orden. Se puede apreciar que en la Tabla (4.8) se forma una clave numérica, dada una configuración de variables básicas y no básicas.

Entonces esta clave numérica se ingresa a la tabla hash y se le asociará con un valor (ID) para cada vértice, como se ve en la Tabla (4.9).

Con esto se garantiza tener el control y la información actualizada de los valores en las configuración de todos los vértices de intersección que se ingresen en la pila de almacenamiento, obteniendo así una lista de las soluciones del sistema que se está resolviendo así como un mayor control sobre la información de todos los vértices intersección.

Tabla 4.8: Ejemplo de formación de clave numérica.

Variables		
variables básicas:	$x_2,\ x_1,\ s_2,\ s_3$	
variables no básicas:	$s_1,\ s_0,\ x_3$	
		Valores numéricos
Variables ordenadas:	$x_1, x_2, x_3, s_0, s_1, s_2, s_3$	$\rightarrow 1, 0.5, 0, 0, 0, 0.5, 1$

Tabla 4.9: Tabla de control para la configuración numérica.

Clave binaria		Clave (ID)		Valores Numericos
				$x_1\ x_2\ x_3\ s_0\ s_1\ s_2\ s_3$
0100010101	$\rightarrow$	1	$\rightarrow$	1, 0.5, 0, 0, 0, 0.5, 1
0101010101	$\rightarrow$	2	$\rightarrow$	1, 0, 2.4, 5, 0, 0, 7.1
$\vdots$		$\vdots$		$\vdots$

4.5. Comentarios finales

Este capítulo se inició con una descripción de la interpretación del problema de la intersección desde un enfoque de PL, lo que permite analizar el problema de la intersección entre dos cuerpos geométricos en una serie de ecuaciones lineales, haciendo que el problema sea más sencillo de resolver a través de las técnicas de PL como lo es el método simplex. Después se presentó la modificación al método Simplex (MSVNB) en algunos de los criterios que rigen al Simplex la cual ayuda a determinar los vértices de intersección entre la hipercaja e hiperplano, pudiendo determinar con certeza los vértices de intersección adyacentes. Para posteriormente presentar una de las técnicas de recorridos de grafos basadas en DFS, que ayudará a recorrer todos los vértices de intersección evitando ciclos o repeticiones entre las adyacencias de los vértices. Y finalmente se mencionan las implementaciones realizadas para llevar el control de los vértices con la finalidad de enlistar todos los vértices solución al sistema con toda la información binaria y numérica de cada uno. Habiendo discutido los detalles de diseño del comportamiento de la modificación al método Simplex (MSVNB) y

la dinámica de cambio que se utiliza en el mismo, en el siguiente capítulo se presentan el algoritmo solución así como un ejemplo del funcionamiento completo.

Capítulo 5

Implementación del algoritmo solución

Este capítulo presenta una descripción de la implementación general del algoritmo solución implementado en este proyecto para la detección de los vértices de intersección entre la hipercaja e hiperplano. Se inicia con algunas definiciones preliminares que necesitan ser mencionadas y que dará lugar a la manera de describir el algoritmo de intersección que se propone. Posteriormente se muestra a detalle cada uno de los pasos propuestos en conjunto con sus respectivos algoritmos y ejemplos ilustrativos de funcionamiento. Finalmente se hacen algunos comentarios finales.

5.1. Implementación del algoritmo solución

En esta sección se comenta el algoritmo solución propuesto para computar las intersecciones entre la hipercaja e hiperplano. Primero hay algunas definiciones preliminares que necesitan ser asentadas, y que darán lugar a la manera en que se describe el algoritmo de intersección, tales como la distancia máxima que debe tener el hiperplano en relación a la hipercaja, la generación de los valores para las variables que conforman las restricciones del problema y la función objetivo.

Por simplicidad, para explicar el algoritmo se considerará el caso donde la gene-

ración de las hipercajas sean unitarias y localizadas en el origen, el hecho de trabajar con hipercajas unitarias no representa ningún problema, ya que esta propuesta esta propuesta está basada en el método Simplex para el cual no existen limitaciones.

En cuanto a los coeficientes que describen el sistema de ecuaciones lineales para el hiperplano se asignan aleatoriamente de manera uniforme entre 0.01 y 20.0 lo cual garantiza la generación de hiperplanos en distintos ángulos y direcciones. También se propone que los hiperplanos sean creados aleatoriamente y se crea un hiperplano por cada hipercaja, tal que el hiperplano no se aleje demasiado de la hipercaja y haya una mayor probabilidad de intersección entre los dos cuerpos geométricos.

Para asegurarse de que el hiperplano no esté demasiado alejado la hipercaja, se toma en cuenta el parámetro de la distancia que existe entre el origen y un punto, en este caso se toma como referencia el punto más alejado del origen que permita al hiperplano estar cerca de la hipercaja.

En el ejemplo de la Figura (5.1) se puede apreciar la distancia desde el origen al punto más alejado de la hipercaja dada, esa distancia es la máxima permitida tal que el hiperplano a una pendiente de 1 en todas sus dimensiones no se aleje demasiado de la hipercaja y haya más probabilidad de que este intersecte.

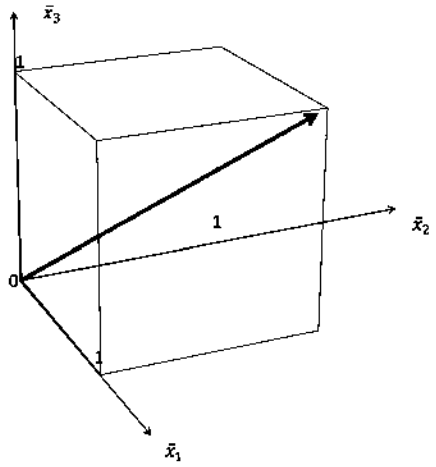


Figura 5.1: Distancia entre el origen y un punto.

Entonces la ecuación descrita para la distancia dadas las características del pro-

blema es la presentada en la Fórmula (5.1).

$$\begin{aligned} D &= \sqrt{\bar{x}_1 + \bar{x}_2 + ... + \bar{x}_i} \\ Donde \quad & i = 1....n \end{aligned} \tag{5.1}$$

A continuación se muestra el algoritmo solución propuesto para este proyecto.

5.2. Algoritmo propuesto.

A continuación se analizará el Algoritmo (4), para la detección de las intersecciones entre una hipercaja y un hiperplano, el cual se compone en la primera parte del método Simplex (MSVNB), descrito en el Capítulo 3, para posteriormente incorporar la modificación al método Simplex (MSVNB) junto con las técnicas del recorrido en profundidad de árboles y las tablas hash para el control de los vértices descritas en el Capítulo 4, quedando de la siguiente manera.

Algoritmo 4: Algoritmo HHI para la búsqueda de las Intersecciones entre la Hipercaja e Hiperplano, o por sus siglas en inglés, Hyperbox and Hyperplane Intersection.

```
1: Nodo ← Simplex _MSVNB(A, c, rhs)
2: Pila.apped(Nodo)
3: Clave ← ConfiguracionBinaria(Nodo)
4: InsertaClaveEnTablaHash(Clave)
5: MarcarVerticeNoVisistado(Clave)
6: mientras Pila != Null hacer
7:    Nodo' ← Pila.pop()
8:    ClaveNumerica ← ConfiguracionNumerica(Nodo')
9:    InsertaClaveEnTablaHash(ClaveNumerica)
10:   MarcarVerticesVisitados(ClaveNumerica)
11:   v ← BuscarVecinos(Nodo')
12:   para i = 1 . . . v hacer
13:      (Nodo'_adyacente) ← EscogeVecinoAdyacente(Nodo')
14:      Clave ← ConfiguracionBinaria(Nodo'_adyacente)
15:      si InsertaClaveEnTablaHash(Clave) No existe entonces
16:         Simplex_MSVNB_Modificado(Nodo'_adyacente)
17:         Pila.apped(Nodo'_adyacente)
18:         MarcarVerticeNoVisitado(Clave)
19:      fin si
20:   fin para
21: fin mientras
22: devolver TablaHash
```

El algoritmo final HHI (por sus siglas en inglés Hyperbox and Hyperplane Intersection) para la solución del problema de intersección entre una hipercaja y un hiperplano comienza recibiendo como parámetros iniciales los datos del sistema; una matriz de las ecuaciones (restricciones) A, la función objetivo Z y el lado derecho de las ecuaciones rhs.

El algoritmo HHI envía estos parámetros iniciales al método Simplex MSVNB descrito en el Capítulo 3, el cual es el encargado de obtener la primera SBF óptima del sistema, es decir, un vértice de intersección entre la hipercaja e hiperplano. Luego se comienza a hacer las operaciones necesarias para visitar a cada uno de los vértices asociados a las soluciones de intersección, visitando a cada vértice una sola vez.

El algoritmo propuesto basado en el recorrido de vértices DFS comienza a desplazarse entre los vértices de intersección que se encuentren en el sistema. Como datos iniciales se necesita la información del vértice de intersección encontrado por el método Simplex MSVNB como se observa en la línea 1.

Una vez que el algoritmo recibe como parámetros de entrada la información del vértice de intersección, en la línea 2 es ingresado a una pila de almacenamiento, donde se irán guardando todos los vértices de intersección que se vayan encontrando durante la búsqueda para que posteriormente sean visitados.

Luego en las líneas 3 y 4 se asigna una clave binaria, basada en las variables básicas y no básicas que contenga el vértice de intersección y es ingresada a una tabla hash para llevar el control de los vértices de intersección que se vayan ingresando con el fin de evitar repeticiones.

Luego en la línea 5 se marca el vértice como no visitado, ya que solo se marcarán como visitados cuando sean extraídos de la pila de almacenamiento y sean analizados para obtener los valores numéricos de las variables básicas y no básicas que contengan para enseguida analizar la relación que tiene con otros vértices adyacentes que puedan ser ingresados a la pila.

Después se inicia un ciclo while, línea 6, que estará encargado de procesar la información que se encuentre dentro de la pila de almacenamiento de los vértices de intersección no visitados, y se detendrá una vez que la pila este completamente vacía.

Dentro del ciclo while, en la línea 7 se procede a extraer un vértice de intersec-

ción, como ya se mencionó anteriormente la pila de almacenamiento implementada en este proyecto utiliza una estructura LIFO, entonces se extraerá al último vértice de intersección que se ingresó.

Cada vez que un vértice de intersección es extraído de la pila, es con la finalidad de visitarlo para obtener su configuración numérica, línea 8, e ingresarla a una tabla de control numérico, línea 9, para así obtener los valores de cada una de las variables básicas y no básicas que lo conforman, para después marcar a ese vértice de intersección como visitado, línea 10, así se evita re-visitar nodos ya visitados.

La tabla de configuración numérica nos da las soluciones al sistema de intersección descrito y ayuda a tener un mejor control en los valores de variables básicas y no básicas que contiene cada vértice de intersección para saber entre qué valores se encuentra definido el sistema y asegurarse de estar siempre sobre el mismo hiperplano.

En la línea 11 se procede a buscar el número de vértices de intersección vecinos que tiene el vértice que acaba de ser extraído de la pila de almacenamiento con el objetivo de realizar un análisis en ese vértice y verificar si existen adyacencias con algún o algunos otros vértices de intersección para posteriormente ingresarlos a la pila de almacenamiento.

En la línea 12 se inicia un ciclo que va desde 1 hasta v, donde v representa los vértices de intersección adyacentes que tenga el vértice extraído que se este analizando, con la finalidad de ingresar a la pila de almacenamiento a los vértices de intersección adyacentes.

En caso de que haya al menos un vértice de intersección vecino en la línea 13 se procede a escoger uno a uno para obtener en la línea 14 su clave única basada en la configuración binaria de sus variables básicas y no básicas.

Después en la línea 15 se verifica que la clave que se asigna al vértice vecino sea distinta de las que ya están en la tabla hash, si la clave es igual a otra entonces se dice que ese vértice de intersección ya ha sido ingresado y seguramente es vecino de algún otro vértice de intersección que ingresó primero. Cabe señalar que no se puede ingresar la misma clave más de una vez a la tabla hash ya que solo se aceptan un elemento por clave, sin repeticiones.

Suponiendo que la clave fue aceptada entonces esta es ingresada a la tabla hash y en la línea 16 se realiza el procesamiento del nuevo vértice de intersección llamando a la

función PivoteoParcial, descrita anteriormente en el Capítulo 3, la cual es utilizada para desplazarse del vértice de intersección origen al vértice de intersección vecino para poder llegar a él y obtener su configuración numérica.

Enseguida en la línea 17 y 18, se procede a ingresar en la pila a todos los vértices vecinos que se encuentren y se marcan como vértices no visitados respectivamente. Después que se termina de ingresar a todos los vértices de intersección adyacentes el ciclo "for" termina en la línea 20.

En la línea 21 se verifica sí el ciclo "while" tiene algún vértice de intersección que falte de extraer en la pila. En caso de que haya más vecinos pendientes este ciclo sigue hasta que todos los vértices del politopo sean visitados una sola vez y se hayan guardado sus configuraciones en las respectivas tablas de control correspondientes.

Finalmente en la línea 22 se devuelve las tablas hash donde se encuentra la lista de todos los vértices de intersección que son la solución al sistema, con su respectiva información binaria y numérica.

El algoritmo propuesto es muy sencillo de entender, y de fácil implementación. A continuación se expondrá un ejemplo sencillo para que se vea gráficamente como trabaja este algoritmo.

5.3. Ejemplo de algoritmo solución.

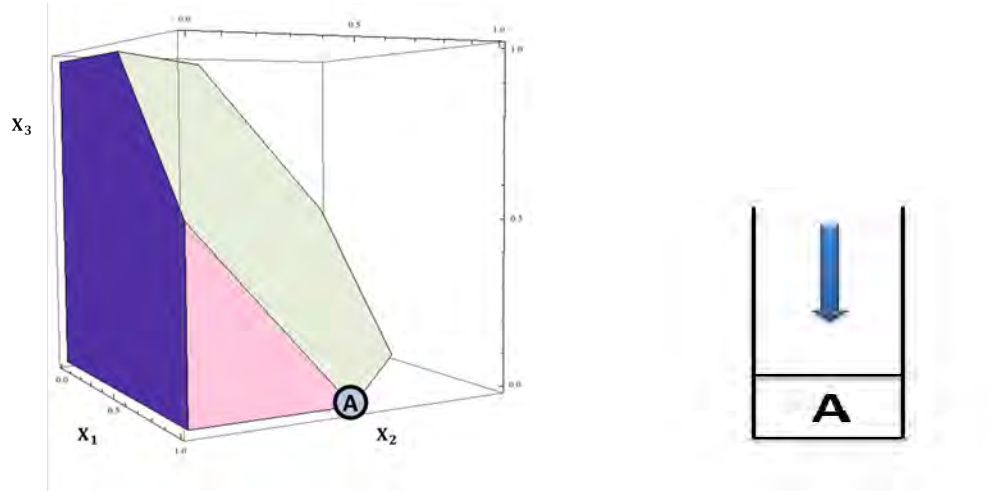
Para una mejor comprensión del algoritmo HHI se propone un pequeño ejemplo en el que se ilustra paso a paso la mecánica del mismo.

Siguiendo el ejemplo mostrado en el Capítulo 4 Sección (4.2) para encontrar las intersecciones entre la hipercaja e hiperplano, se tiene un sistema lineal en el cual se describe a una hipercaja de tres dimensiones que es intersectado por un hiperplano. Entonces se asumirá la existencia de un vértice de intersección que es la solución inicial trivial que es arrojada por el método Simplex (MSVNB), basados en tabla 4.2.

En la Figura (5.2) tenemos la primera solución inicial que arroja el método Simplex MSVNB, a la cual se le llama el vértice A. También se dibuja la pila que se utiliza para verificar cuales vértices faltan por procesar, recordando que se utiliza una estructura de tipo

LIFO.

Siguiendo el algoritmo HHI se inicializa la pila ingresando el primer vértice de intersección como se ve en la Figura (5.2(b)), y se procede a asignarle una clave binaria sujeta a la configuración de las variables que el mismo vértice tenga. Se ingresa a las tablas de control la clave binaria y se procede a marcar al vértice como no visitado como se aprecia en la Tabla (5.1).



(a) ***Ilustración de la primer intersección entre la hipercaja he hiperplano encontrada.*** **(b)** ***Primer vértice de intersección ingresando a la pila.***

Figura 5.2: Ejemplo de recorrido de vértices. Primer vértice de intersección ingresado a la pila de almacenamiento.

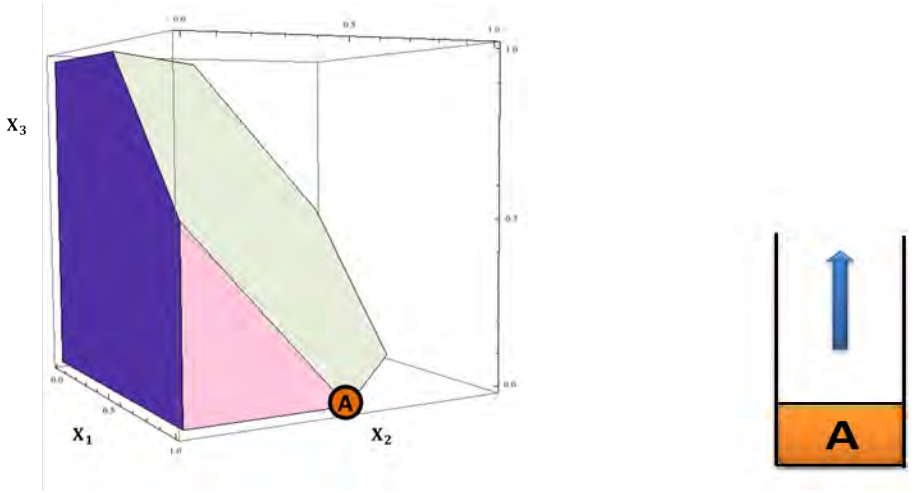
Tabla 5.1: Ejemplo de llenado de tablas de control. Ingresa primer vértice de intersección a las tablas de control.

Clave binaria		Valor (ID)		Visitados		Configuración Numérica
1 1 0 0 0 1 1	→	1 (A)	→	0	→	Null
⋮		⋮		⋮		⋮

Una vez que se conoce cual es el vértice de intersección inicial y se han inicializado las tablas de control entonces la pila de vértices de intersección comenzará a realizar las operaciones para visitar todos los vértices de intersección que se vayan recolectando en ella.

El algoritmo HHI procede a verificar si existe al menos un vértice de intersección dentro de la pila de almacenamiento, si existe al menos uno se procede a extraer el último vértice que se ha ingresado.

Como se ve en la Figura (5.3), se extrae de la pila de almacenamiento el nodo A. Cada vez que un vértice de intersección es extraído de la pila se ejecutará la extracción de la información numérica que contenga el vértice, insertándola en las tablas de control y también se marcara al vértice como visitado. Esto se puede observar en la Tabla (5.2).



(a) ***Ilustración visitando al vértice de intersección*** A***.***

(b) ***Extrayendo de la pila de almacenamiento*** A***.***

Figura 5.3: Ejemplo de recorrido de vértices. Visitando a nodo A.

Tabla 5.2: Ejemplo de llenado de tablas de control. Extracción del primer vértice de la pila.

Clave binaria		Valor (ID)		Visitados		Configuración Numérica
1 1 0 0 0 1 1	→	1 (A)	→	1	→	1 0.5 0 0 0 0.5 1
⋮		⋮		⋮		⋮

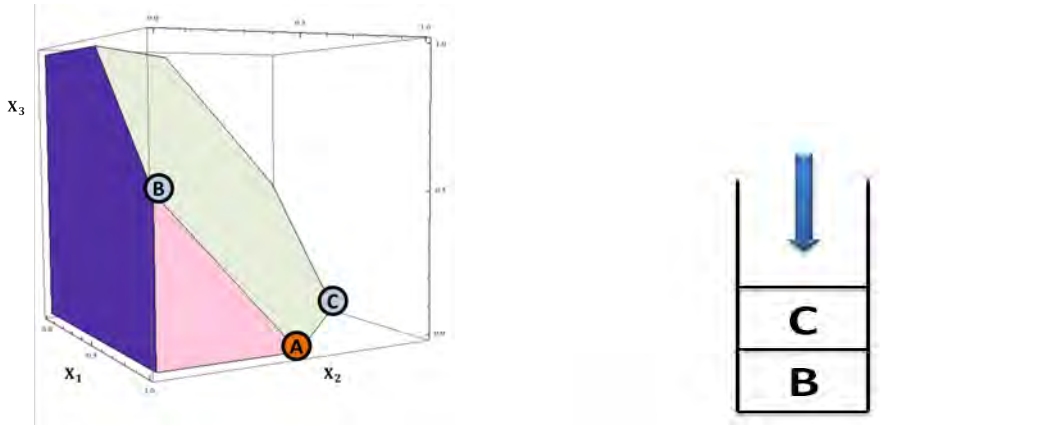
Una vez que se termina de analizar el nodo A, se procede a buscar si el mismo vértice de intersección extraído de la pila de almacenamiento tiene algún vértice de intersección adyacente al cual se pueda tener acceso a través de él. Si al menos existe un vértice

de intersección vecino se procederá a encontrar su configuración para ingresarlo en la pila de almacenamiento.

En el ejemplo observamos que el nodo A presenta dos vértices vecinos, estos se pueden consultar en el Capítulo 4 en la Tabla (4.2) donde el primer vértice de intersección vecino que se presenta está asociado a la variable no básica s_1 y el segundo vértice vecino está asociado a la variable no básica x_3.

Por simplicidad se asigna una letra para cada vértice de intersección adyacente encontrado, el nodo B representa la decisión de ingresar en la base a la variable s_1 y el nodo C representa la decisión de ingresar en la base a la variable x_3.

Entonces se verifica cada vértice de intersección adyacente en orden de izquierda a derecha como se muestra en la Tabla (4.2) y se ingresa a la pila de almacenamiento uno a uno. El nodo B se inserta primero y posteriormente el nodo C. Gráficamente esto se puede apreciar en la Figura (5.4).



(a) ***Ilustración de los vértices de intersección adyacentes al nodo*** A***.***

(b) ***Ingresando vétices adyacentes al nodo*** A

Figura 5.4: Ejemplo de recorrido de vértices. Ingresando vértices de intersección adyacentes al nodo A.

Para que cualquier vértice pueda ser ingresado, primero se extrae la clave binaria la cual es tomada en base a las variables básicas y no básicas que contenga, se comprueba que esta clave no exista en las tablas de control. Si la clave no está repetida, se ingresa al vértice B en la pila para realizar el pivoteo parcial y lo mismo sucede con el vértice vecino

C.

El procesamiento de los vértices de intersección vecinos ingresados a la pila se lleva a cabo mediante un pivoteo parcial y finalmente se marcan como no visitados y se actualizan las tablas de control como se muestra en la Tabla (5.3).

Tabla 5.3: Ejemplo de llenado de tablas de control. Ingreso de los vértices de intersección B y C.

Clave binaria		Valor (ID)		Visitados		Configuración Númerica
1 1 0 0 0 1 1	→	1 (A)	→	1	→	1 0.5 0 0 0 0.5 1
0 0 1 1 0 1 0	→	2 (B)	→	0	→	Null
0 1 0 1 1 0 0	→	3 (C)	→	0	→	Null

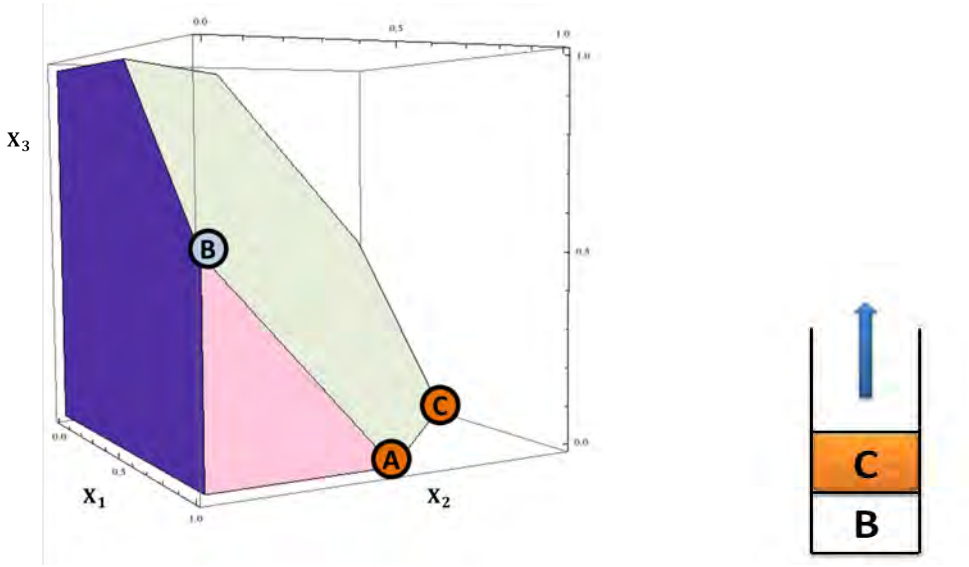
Una vez ingresados todos los vértices de intersección vecinos del nodo A, se procede a repetir todo el procedimiento de nuevo hasta que no quede ningún vértice de intersección dentro de la pila de almacenamiento.

En la Figura (5.5) se observa que el siguiente nodo en salir de la pila es el vértice C, y se repite el procedimiento para el vértice C, extrayendo la configuración numérica e insertándola en la tabla de control numérico, cambiando el estado del mismo de no visitado a visitado y se actualizan las tablas correspondientes como se muestra en la Tabla (5.4).

Tabla 5.4: Ejemplo de llenado de tablas de control. Extracción del vértice C de la pila.

Clave binaria		Valor (ID)		Visitados		Configuración Numérica
1 1 0 0 0 1 1	→	1 (A)	→	1	→	1 0.5 0 0 0 0.5 1
0 0 1 1 0 1 0	→	2 (B)	→	0	→	Null
0 1 0 1 1 0 0	→	3 (C)	→	1	→	1 0 0.5 0 0 1 0.5

Es necesario verificar si el nodo C, tiene algún vértice de intersección adyacente al cual se pueda acceder desde la configuración del mismo. Esto se puede verificar en la Tabla (4.3) del Capítulo 4 donde se observa que existen dos vértices de intersección vecinos, los cuales están asociados a una variable no básica cada uno. Por simplicidad se menciona que la variable básica asociada a s_2 será el nodo D y que la variable asociada a x_3 será el nodo



(a) ***Ilustración visitando a vértice de intersección*** C**.**

(b) ***Extrayendo de la pila de almacenamiento a vértice*** C

Figura 5.5: Ejemplo de recorrido de vértices. Visitando a nodo C

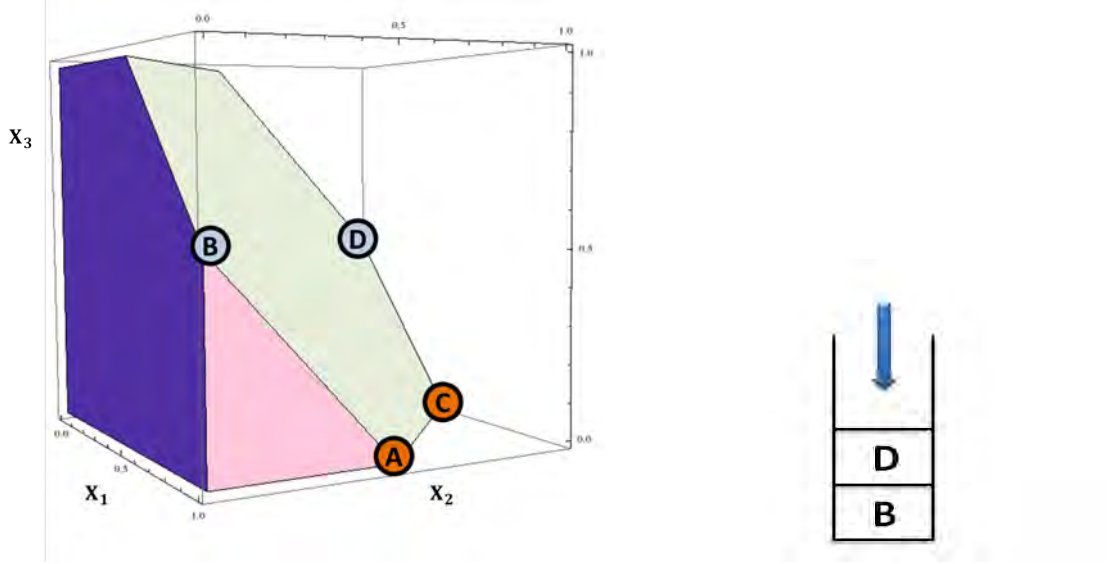
A.

Gráficamente en la Figura (5.6) se observa que el nodo C contiene dos vecinos adyacentes A y D, sin embargo la variable no básica asociada al vértice de intersección A, es la que acaba de salir de la base y su reingreso daría como resultado la misma configuración anterior que se presentó en el nodo A, por lo que se estaría regresando a la configuración anterior y eso sería una repetición y se podría caer en ciclos infinitos.

Por lo que el algoritmo HHI procede a ingresar a todos los vecinos adyacentes al vértice de intersección C que tengan una clave binaria no repetida. El resultado se ve en la Figura (5.6), donde se muestra que únicamente se agregó el nuevo vértice de intersección D y se actualizan las tablas de control correspondientes, como se observa en la Tabla (5.5).

Posteriormente el proceso se repite, extrayendo de la pila de almacenamiento al último nodo ingresado que en este ejemplo sería el nodo D. Este procedimiento se repite hasta que la pila de almacenamiento quede completamente vacía, lo que querrá decir que ya no hay más vértices de intersección a los que se pueda acceder.

Una vez que la pila de almacenamiento está completamente vacía, se sabe que ya



(a) ***Ilustración buscando vértices adyacentes de*** C**.** **(b)** ***Ingresando vértices adyacentes a nodo*** C**.**

Figura 5.6: Ejemplo de recorrido de vértices de intersección. Ingresando vértices adyacentes a nodo C.

no hay más vértices de intersección que visitar, ya que el mismo algoritmo de recorrido de grafos utilizados se ha encargado de revisar todas las adyacencias de los vértices que van conformando la intersección.

El método Simplex MSVNB modificado únicamente puede tener acceso a los vértices vecinos de un nodo dado, pues trabaja ingresando una sola variable a la base y extrayendo una variable de la base, lo que se traduce como realizar un solo movimiento entre los vértices.

Por ejemplo en la Figura (5.6) el vértice A tiene como vértices adyacentes a los nodos B, C esto quiere decir que en el Tableau perteneciente a el vértice A tiene una configuración donde hay únicamente dos variables no básicas que tienen un 0 en el renglón $c_j - z_j$, las cuales pertenecerían a B y A respectivamente, por lo que el método Simplex MSVNB modificado solo podría ingresar a la base a uno de estos dos vértices, de tal manera que es imposible tener acceso al vértice D desde el vértice A.

Por lo anterior se llega a la conclusión que todos los vértices de intersecciones

Tabla 5.5: Ejemplo de llenado de tablas de control. Ingresando vértices de intersección D.

Clave binaria		Valor (ID)		Visitados		Configuración Numérica
1 1 0 0 0 1 1	→	1 (A)	→	1	→	1 0.5 0 0 0 0.5 1
0 0 1 1 0 1 0	→	2 (B)	→	0	→	Null
0 1 0 1 1 0 0	→	3 (C)	→	1	→	1 0 0.5 0 0 1 0.5
0 0 0 0 0 0 0	→	4 (D)	→	0	→	Null

entre la hipercaja e hiperplano han sido visitados. En este caso en específico se trató de un problema con seis intersecciones y se puede observar que todos fueron visitados una sola vez.

Cabe señalar que esta técnica puede utilizarse para problemas de n variables, esto significa que se puede trabajar con cualquier número de dimensiones que sean necesarias. En el siguiente capítulo se expondrán algunos de los resultados que se obtuvieron con distintas dimensiones.

5.4. Comentarios finales

En este capítulo se expusieron las técnicas utilizadas para realizar el recorrido en los vértices de intersección entre la hipercaja e hiperplano. Se inició con la definición de algunos términos comunes y posteriormente se expuso a detalle cada uno de los pasos propuestos en conjunto con sus respectivos algoritmos y ejemplos ilustrativos de funcionamiento. Se presentó una descripción del esquema propuesto en este trabajo para realizar el control de los vértices mediante el recorrido de los mismos. Habiendo discutido los detalles de diseño e implementación del algoritmo propuesto, en el siguiente capítulo se presentan los resultados de los experimentos realizados.

Capítulo 6

Pruebas y Resultados

Hasta el momento se ha descrito la metodología desarrollada para realizar el cálculo de las intersecciones entre una hipercaja e hiperplano. En este capítulo se describen las pruebas desarrolladas para comprobar el desempeño del sistema, obteniendo resultados cuantitativos que permiten apreciar el comportamiento del mismo.

6.1. Parámetros para generar los conjuntos de datos

En el presente capítulo verifica el funcionamiento del algoritmo propuesto, y se comprueba el desempeño del sistema, con la finalidad de obtener resultados que permitan apreciar el comportamiento del mismo. Para esto se desarrollan pruebas entre una hipercaja y un hiperplano, se describen algunos parámetros que deben tomarse en consideración para su práctica, de tal manera que validen el proyecto como resultado del problema definido en el Capítulo 1.

La implementación de este proyecto se realizó en el lenguaje de programación Python, y para la elaboración de las pruebas, se tomó en cuenta varios factores, que ya fueron comentados en párrafos precedentes, tales como: la distancia máxima que debe tener el hiperplano en relación a la hipercaja, la generación de los valores para las restricciones del problema, la función objetivo y la verificación del número de soluciones resultantes que no sobrepase la cota máxima, para cada prueba se generan muestras de 100 hipercajas unitarias y 100 hiperplanos aleatorios por cada dimensión, que van desde la tercera hasta

vigésima dimensión.

Como se mencionó anteriormente, todos las hipercajas utilizadas en las pruebas son unitarias y posicionadas en el origen con la finalidad de simplificar los resultados y poder hacer gráficos comparativos, mientras las variables que definen al hiperplano (restricción y función objetivo Z) son valores aleatorios entre 0.01 y 20.0 con la finalidad de obtener una mayor diversidad en ángulos y formas que tome el hiperplano.

En la primera prueba, se cuantifica el tiempo de procesamiento, el número de soluciones encontradas y se grafican los resultados, para observar el comportamiento del sistema.

En la segunda prueba, se cuantifica el promedio del tiempo de ejecución que le toma al algoritmo HHI encontrar un vértice de intersección solución por cada dimensión para posteriormente graficar los resultados y realizar comparaciones.

En la tercera prueba, se va a muestra el tiempo de ejecución de la función Pivoteo Parcial utilizada en el método Simplex (MSVNB), descrita en el capítulo 3, para verificar el tiempo que le toma al método realizar las operaciones matriciales 1 millón de veces por cada dimensión.

6.2. Pruebas y resultados

En los párrafos precedentes se asentaron los factores para la elaboración de pruebas y que resultados arrojan. A continuación se presentan las pruebas descritas.

Para la primera prueba por cada dimensión se puede observar el comportamiento de los siguientes parámetros: el número de vértices de intersección que son soluciones del sistema y el tiempo de procesamiento.

En la Figura (6.1), se muestran los resultados para una dimensión de $n = 3$, donde se observa que el mayor número de vértices soluciones encontradas fue de seis, mientras que el menor número de soluciones encontradas fue de tres, con un tiempo máximo de ejecución de 0.005.9 segundos. Estos valores brindan una idea del comportamiento que tiene el algoritmo solución al encontrar distintos número de soluciones, donde para resolver un problema con 3 vértices de intersección tardará entre 0.001 seg hasta 0.002 segundos

mientras que para resolver un problema con 4 vértices de intersección tardará entre 0.0015 segundos hasta 0.0058 segundos y así sucesivamente entre los diferentes números de vértices solución que pueda tener un sistema. Se hizo este procedimiento para todas las dimensiones desde $n = 3$ hasta $n = 20$, los resultados se pueden observar gráficamente en el Apéndice (A), donde se muestran los valores del tiempo de ejecución y número de vértices de intersección soluciones encontrados por cada dimensión.

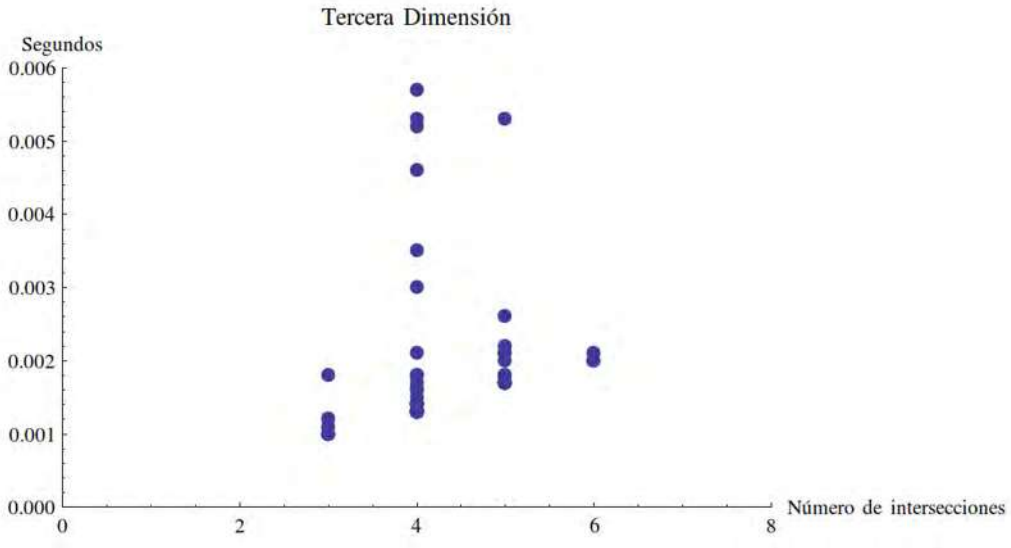


Figura 6.1: Resultados obtenidos en la tercera dimensión.

Para las dimensiones de $n = 19$ y $n = 20$ en las Figuras (6.2) y (6.3) se puede observar que hay un rango de vértices de intersección soluciones muy grande respecto a las dimensiones anteriores, pues conforme aumenta la dimensión aumenta la posibilidad de encontrar un mayor número de vértices de intersección, se aprecia un comportamiento cuadrático en el que la dimensión 19 presenta más de $100,000$ vértices de intersección solución con un tiempo de ejecución de la prueba de casi $10,000$ segundos mientras que para la dimensión 20 se observan millones de vértices solución encontrados, siendo el número más alto un aproximado de $1,600,000$ vértices de intersección con un tiempo de ejecución de la prueba de $45,000$ segundos equivalente a 12.50 horas aproximadamente.

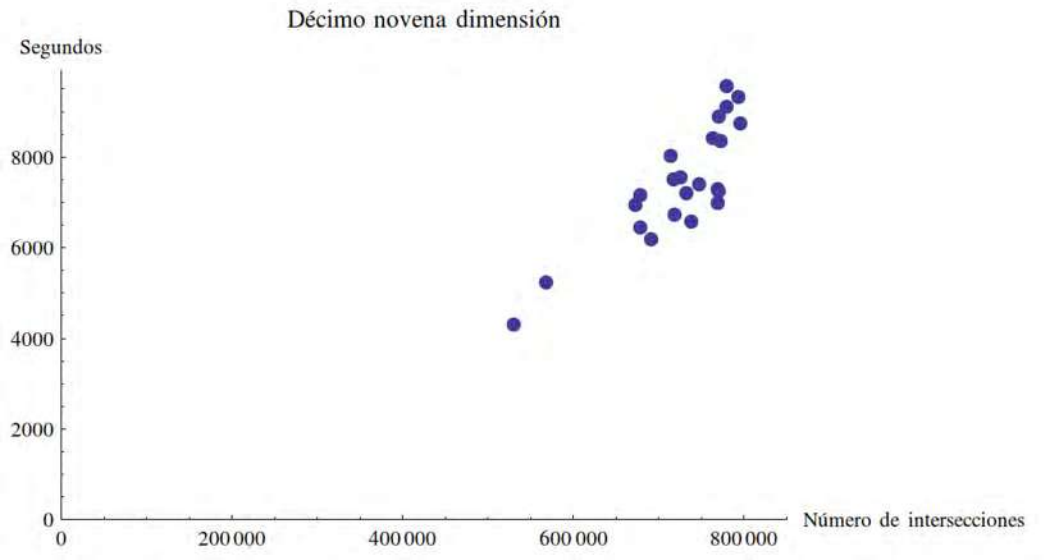


Figura 6.2: Resultados obtenidos en la décimo novena dimensión.

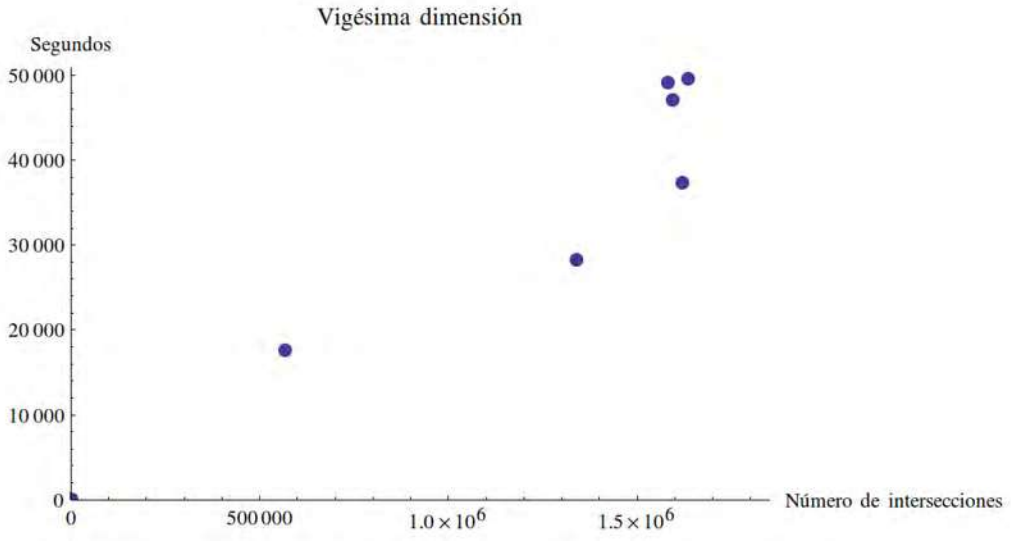


Figura 6.3: Resultados obtenidos en la vigésima dimensión.

Ahora bien, se presentan todos los resultados obtenidos en las 20 dimensiones divididos en dos Figuras, (6.4) y (6.5) en las cuales se observa una muestra aproximada de 100 pruebas por cada dimensión, desde $n = 3$ hasta $n = 20$, en cada una de las cuales se contabilizó el número de soluciones y el tiempo de ejecución.

En conjunto se puede observar que desde $n = 3$ hasta $n = 17$ el número de vértices de intersección encontrados aumenta al igual que el tiempo de ejecución sin embargo, este incremento no es muy elevado en comparación con el tiempo de ejecución que se presenta a partir de la dimensión 18 y esto se debe a que el número de soluciones que se pueden encontrar aumenta de manera significativa, se pueden encontrar hasta casi 2 millones de vértices de intersección siendo el número más alto para una dimensión 20 un aproximado de $1,847,560$ vértices, según las fórmulas presentadas en el Capítulo 2 de Asweel y Oneil, lo que implicaría un tiempo de procesamiento mayor a 12 horas. Entonces se concluye que al incrementar el número de la dimensión, el rango de vértices solución que se pueden encontrar aumenta y este aumento en el número de vértices de intersección tiene un límite, el cual no sobrepasa nunca la cota máxima según las fórmulas presentadas en el Capítulo 2 de Asweel y Oneil, también se puede observar que el tiempo de procesamiento incrementa en cada dimensión de acuerdo al número de vértices de intersección que se tengan que procesar, esto es debido a que al aumentar el número de vértices que se tienen en el sistema el número de operaciones incrementa y las necesidades del almacenamiento de memoria incrementan.

El incremento en el tiempo que tarda en encontrar las soluciones para las distintas dimensiones se refleja en un comportamiento cuadrático, pues el tamaño de la matriz aumenta entre cada dimensión y en consecuencia el número de operaciones matriciales incrementara.

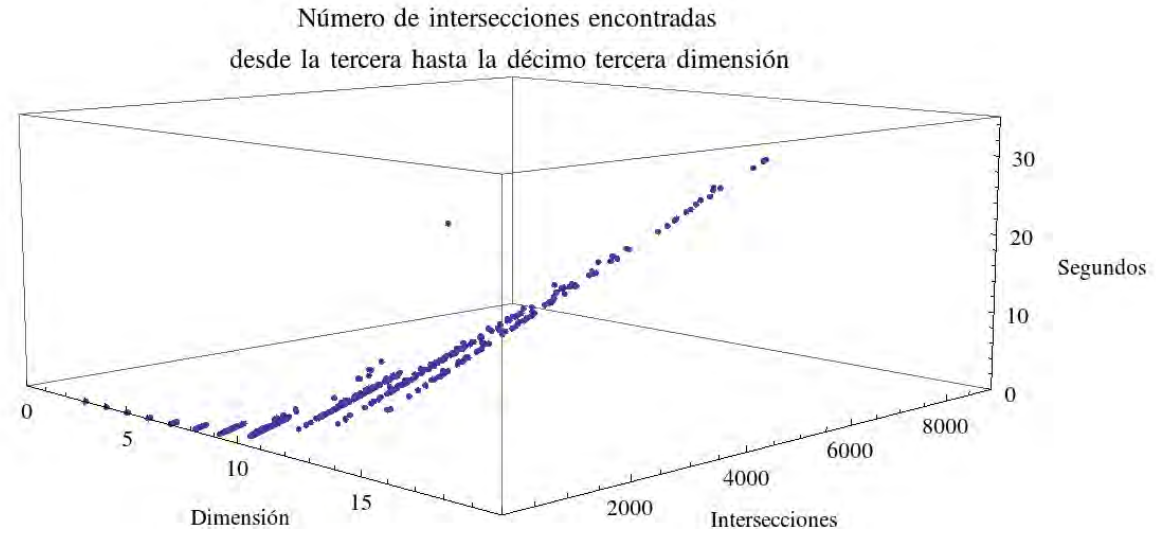


Figura 6.4: Gráfica del número de vértices solución y el tiempo de procesamiento por dimensión abarcando desde la tercera hasta la décimo quinta dimensión.

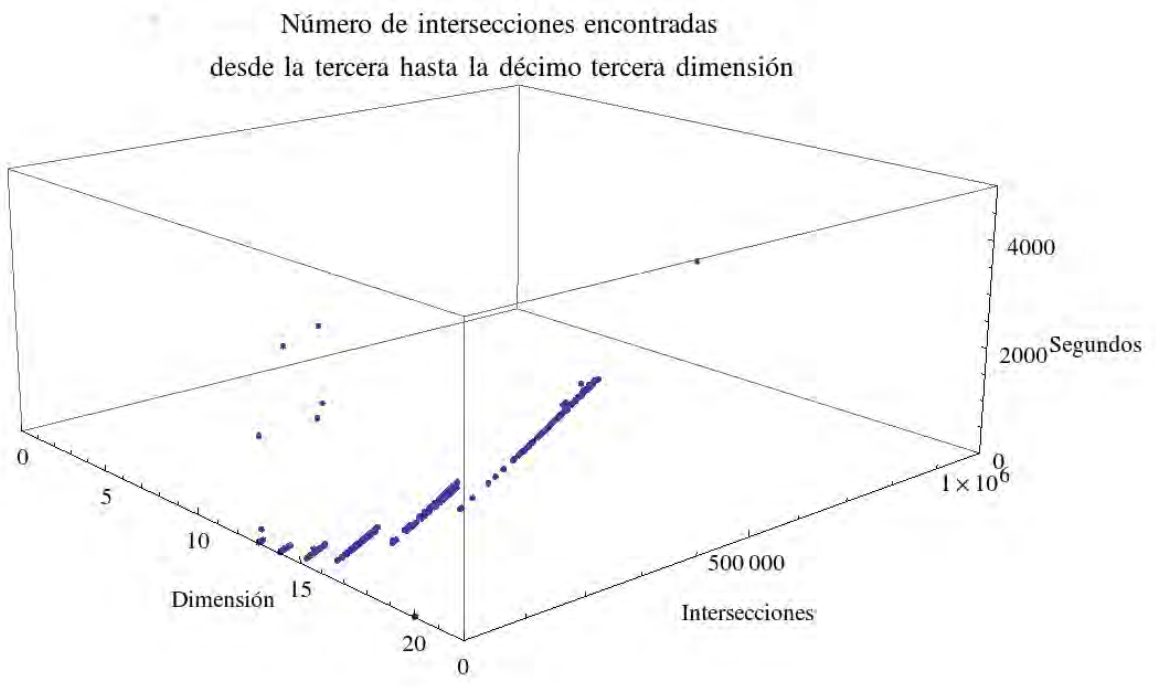


Figura 6.5: Gráfica del número de vértices solución y el tiempo de procesamiento por dimensión abarcando desde la décimo sexta hasta la vigésima dimensión.

Entonces el tiempo de ejecución incrementa conforme se aumenta de dimensión ya que el número de soluciones que se pueden encontrar puede ser mucho mayor dependiendo de las características del sistema. A continuación, se exponen los resultados obtenidos en cada una de las dimensiones, donde se grafica el promedio del tiempo de ejecución que le toma al algoritmo encontrar un vértice de intersección solución en cada dimensión.

En la Figura (6.6) se observa que en la tercera dimensión, el tiempo promedio de ejecución para encontrar un vértice de intersección solución es de 0.00041 segundos, en tanto para la cuarta dimensión es de 0.00056 segundos, y para la quinta dimensión es de 0.00075 segundos de lo cual se obtiene que entre la tercera y la cuarta dimensión hay una diferencia solamente de 0.00015 segundos mientras que entre la cuarta y la quinta hay una diferencia solamente de 0.00019 segundos. Ello hace evidente que el tiempo para encontrar un solo vértice de intersección aumenta dependiendo de la dimensión con la que se esté experimentando, pues al aumentar la dimensión las operaciones matriciales aumentan.

Ese aumento se ve reflejado poco a poco conforme se aumenta la dimensión; así, en base a dicha consideración a partir de la dimensión 18 el tiempo promedio de ejecución se eleva significativamente. Para la dimensión 18 el tiempo promedio de ejecución es de 0.00622 segundos, y para una dimensión 19 es de 0.00904 segundos mientras que para una dimensión 20 es de 0.02105 segundos.

Tomando en cuenta los resultados obtenidos en la Figura (6.6), se podría calcular el tiempo de ejecución aproximado que tardaría para cualquier problema de intersección. Por ejemplo si se plantea un problema para la dimensión 20 en el cual el mayor número de vértices de intersección es de $1,847,560$, según las Fórmulas de (2.2) y (2.5) de Oneil y Ahlswede respectivamente, el tiempo de ejecución que tardaría en encontrar todos los vértices de intersección sería de $38,891.138$ segundos equivalente a 10 horas aproximadamente. Sí se calcula este mismo ejemplo pero en una dimensión 4 el tiempo que tardaría en encontrar los $1,847,560$ vértices de intersección sería de 1034.6336 segundos lo que es equivalente a 17.244 minutos. Esto quiere decir que efectivamente al aumentar la dimensión se aumenta el número de operaciones y por ende el tiempo de ejecución para resolver un problema. Cabe recordar que el máximo número de intersecciones que se pueden encontrar entre una hipercaja y un hipercubo es de 12 vértices.

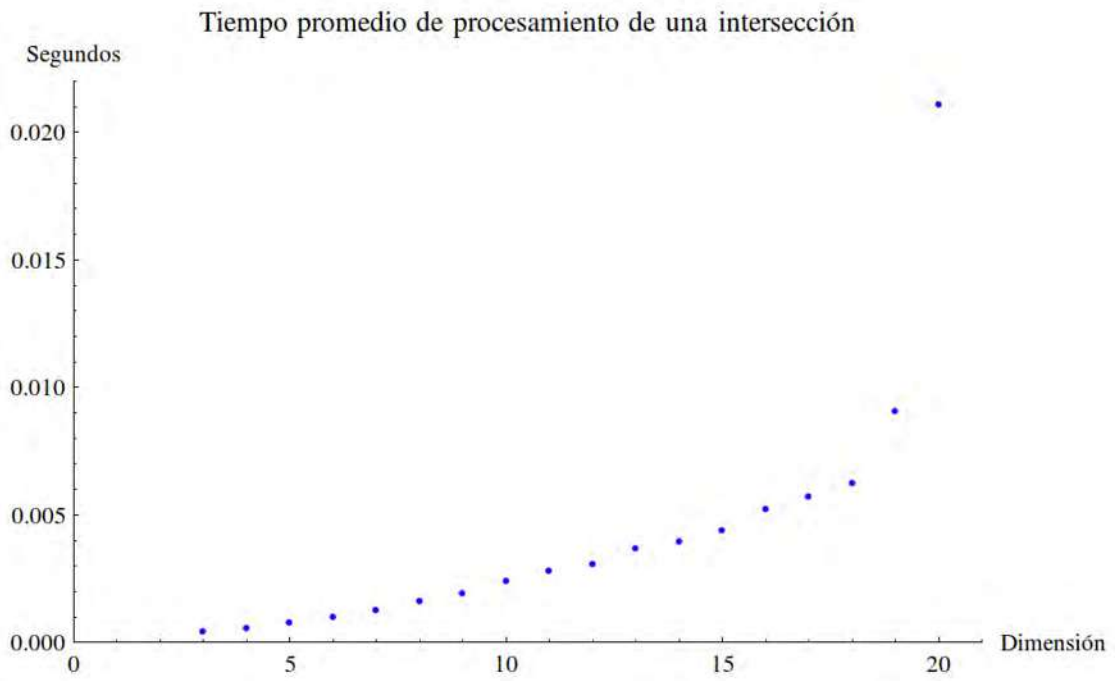


Figura 6.6: Tiempo promedio para encontrar un vértice de solución por dimensión.

En base a tales razonamientos se puede concluir que el tiempo de ejecución crece conforme se eleva la dimensión con la cual se esté trabajando, de tal manera que, el tiempo de ejecución para un problema de intersección entre una hipercaja y un hiperplano dependerá de la dimensión a tratar y del número de vértices solución que se tengan que procesar.

Pero este incremento en el tiempo se debe a varios factores como lo son el aumento en las operaciones matriciales debido a una dimensión más grande, al aumento en el consumo de la memoria y al número de vértices solución que se encuentren en el sistema.

Para verificar el comportamiento del sistema se realizaron pruebas donde se grafica el promedio del tiempo de ejecución de la función Pivoteo Parcial, esto con la finalidad de observar si hay un aumento en el tiempo significativo que se deba al número de operaciones matriciales que se realizan.

Los resultados obtenidos en cada una de las dimensiones se muestran en la Figura (6.7), donde se graficó el promedio del tiempo de ejecución de la función del Pivoteo Parcial,

realizando el cálculo un millón de veces por cada dimensión.

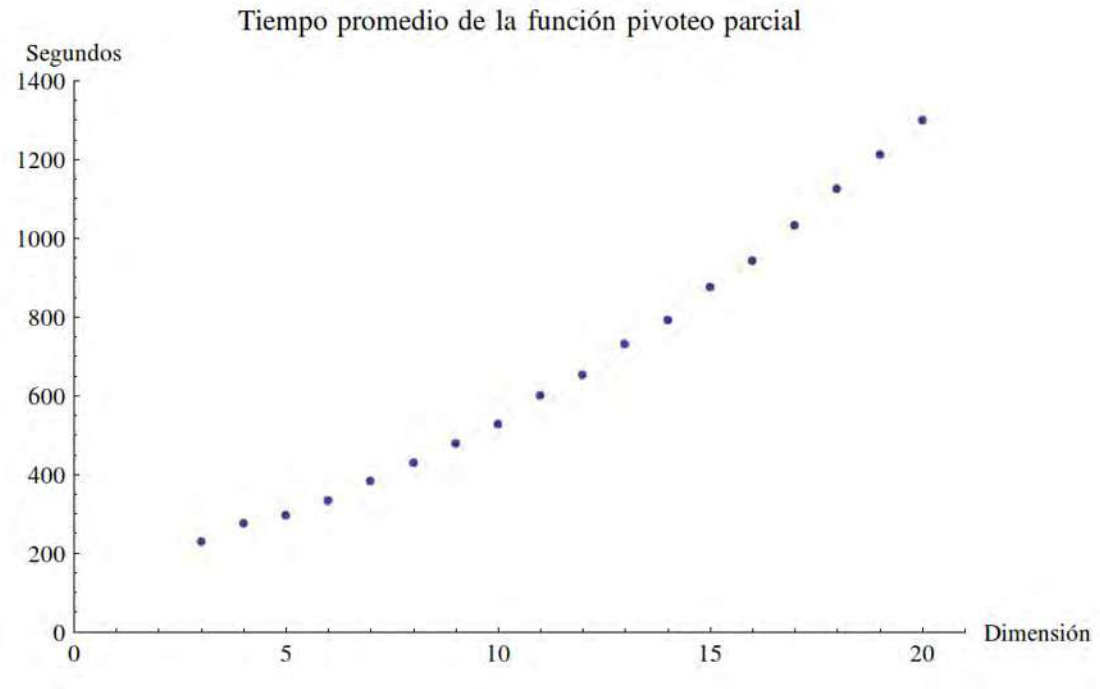


Figura 6.7: Tiempo promedio para encontrar pivoteo parcial por dimensión.

El comportamiento que se presenta en la Figura (6.7) crece de forma cuadrática, esto quiere decir que, si la función pivoteo parcial se necesitará hacer 1 millón de veces para una dimensión 20 el proceso tardaría tan solo 1300 segundos equivalentes a 21 minutos.

Sin embargo como se mencionó anteriormente en las pruebas descritas en la Figura (6.3) para encontrar $1,600,000$ vértices de intersección se requiere un aproximado de 12 horas, lo cual quiere decir que conforme crece el número de operaciones matriciales, aumentan los recursos de la memoria para la administración de la pila, y comienza a degradarse el sistema mostrando dificultades para el manejo de la memoria.

Las ventajas del algoritmo propuesto en relación a otras propuestas por Carlos Lara *et al.* [Lara09] y Rezk Salama *et al.* [Rezk-Salama05], es la eficiencia con la que se desempeña, ya que no importan los límites de la hipercaja ni los coeficientes que describen al hiperplano ya que da total libertad para procesar cualquier problema de intersección entre los dos cuerpos geométricos.

A diferencia del "Naive Algorithm" el algoritmo HHI no procesa todas las aristas del sistema, únicamente se basa en encontrar aquellos vértices que cumplen con los criterios de optimalidad que se proponen a diferencia del método Simplex tradicional. Por tal motivo el "Naive Algorithm" requiere un tiempo de procesamiento mucho mayor, considerando que el número de aristas es siempre mayor al número total de soluciones que se pueden encontrar en un problema de intersección entre una hipercaja y un hiperplano, lo que implica un número innecesario de operaciones en comparación con el algoritmo propuesto.

Además, se podrían encontrar algunas deficiencias al aplicar el "Naive Algorithm", por ejemplo en la Figura (6.8) se presenta el caso donde el hiperplano es paralelo a una de las caras de la hipercaja.

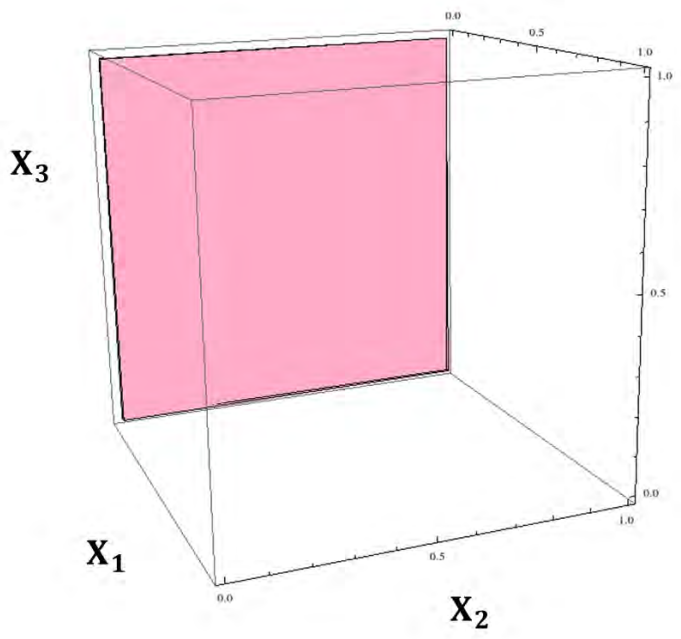


Figura 6.8: Ejemplo de intersección entre una hipercaja y un hiperplano en una sola cara de la hipercaja.

En base a tal ejemplo, al realizar el procesamiento con el "Naive Algorithm" o con el algoritmo presentado por Carlos *et al.* [Lara09] sobre las aristas de la hipercaja que

se encuentran sobre el hiperplano, se les encontrará un número infinito de soluciones, en cambio, el algoritmo propuesto únicamente dará como resultado los 4 vértices de intersección que son esquinas con la hipercaja.

En conclusión, tanto el "Naive Algorithm" como el utilizado en [Lara09] no pueden generalizarse para cualquier problema de intersección entre la hipercaja y un hiperplano, pues fallaría cuando el hiperplano esté sobre una de las caras de la hipercaja. En cambio, el algoritmo HHI no se concentra en un problema en particular si no que funciona para cualquier caso en general, el tamaño de la hipercaja así como la inclinación que presente el hiperplano no generaría ningún problema.

6.3. Comentarios finales

En el capítulo 6 se presentaron algunos resultados de los experimentos realizados. Se definieron los parámetros utilizados y la manera en que afectan el comportamiento del algoritmo propuesto. Posteriormente, se expusieron las pruebas, así como los resultados obtenidos a través de la experimentación. De tal manera que con los resultados arribados en este proyecto, el enfoque del siguiente capítulo será el discutir las conclusiones generales y particulares a las que se arribó y los posibles trabajos futuros que pueden surgir del presente proyecto.

Capítulo 7

Conclusiones

7.1. Conclusiones Generales

El proyecto desarrolla una metodología basada en herramientas de PL, específicamente, una modificación al método Simplex, con la finalidad de determinar los vértices de intersección entre una hipercaja y un hiperplano.

Lo anterior es porque al tratar de resolver el problema de intersección entre dos cuerpos geométricos, se facilita la búsqueda de los vértices solución ya que el sistema es descrito a través de una serie de desigualdades lineales que se pueden resolver mediante el método Simplex y las técnicas aplicadas de recorrido de grafos basadas en DFS.

El método Simplex (MSVNB) utilizado en este proyecto mejora varios aspectos; se reducen tanto el número de operaciones matriciales como el espacio de memoria utilizado, por ende se logra identificar un primer vértice de intersección entre los dos cuerpos geométricos de una manera sencilla y en menor tiempo de procesamiento.

Con respecto al método propuesto HHI, este parte de un primer vértice de intersección que es determinado por el método Simplex (MSVNB), a partir del cual mediante el método Simplex modificado se identifican todos los demás vértices de intersección restantes para que finalmente, basados en un algoritmo de recorrido de grafos se enlistan todos los vértices encontrados en una tabla hash con sus respectivas configuraciones binarias y numéricas.

7.1.1. Conclusiones Particulares

Una conclusión particular de este proyecto, es que convierte satisfactoriamente el problema de encontrar las intersecciones entre una hipercaja y un hiperplano en un problema de PL.

Esta conversión hace que el problema sea modelado a través de ecuaciones lineales donde el hiperplano está representado como la función objetivo y las restricciones están en función de los datos de la hipercaja y también del hiperplano, lo que lo hace más sencillo de resolver.

La utilización del método Simplex (MSVNB) basado únicamente en las columnas no básicas, ayuda a que se realicen menos operaciones, ahorrando espacio en memoria al dejar de almacenar información innecesaria además de que no afecta el comportamiento de dicho método ni los resultados, esto debido a que únicamente se deja de recalcular la matriz identidad que genera el método Simplex en cada iteración. Es importante remarcar que no se compara la propuesta con otros enfoques para resolver problemas de PL basados en punto interior.

Una vez encontrado el primer vértice de intersección se hace la modificación en las reglas por las que se rige el método Simplex con la finalidad de poder determinar los multiples vértices de intersección que son solución al sistema. La modificación al método Simplex (MSVNB) garantiza que: siempre se va a estar buscando únicamente en los vértices de intersección entre la hipercaja y el hipercubo. La modificación al método Simplex radica en modificar dos propiedades, las cuales son el criterio de optimalidad y de ganancia. El criterio de optimalidad modificado, se aplica para determinar entre las variables no básicas una que entre a la base, eligiendo la columna que tenga el coeficiente cero en el renglón "$c_j - z_j$"del tableau para asegurar que se visite un vértice adyacente que sea intersección. El criterio de ganancia modificado consiste en moverse a través del politopo sin mejorar ni empeorar la evaluación de la función objetivo Z.

Además, se implementó satisfactoriamente un algoritmo de eliminación basado en las técnicas de recorridos de DFS para recorrer todos los vértices solución, lo cual permite identificar el área donde se encuentran las soluciones óptimas factibles al problema de la

intersección.

Las tablas de control de vértices basadas en tablas hash, fueron implementadas eficazmente ya que generan las listas de vértices que son soluciones, asociados a las intersecciones entre los dos cuerpos geométricos de una manera óptima y de acuerdo a las necesidades del problema.

De esta manera, se llega exitosamente al objetivo principal, el cual consiste en identificar los puntos de intersección entre una hipercaja y un hiperplano de n dimensiones aplicando métodos sencillos y bastante conocidos, como lo son el método Simplex y las técnicas de recorridos de vértices.

7.2. Trabajos Futuros

Las propuestas para continuar con la mejora del proyecto se enuncian a continuación.

1. Modificar el método Simplex, agregando técnicas como las utilizadas en el Simplex dual.

2. Implementar otras técnicas de recorrido de árboles como árboles de expansión, búsqueda a lo ancho, etc. Con la finalidad de obtener una comparativa y verificar cual es el mejor recorrido para los vértices.

3. Flexibilizar el algoritmo para encontrar las intersecciones entre una hipercaja y n hiperplanos.

4. Generalizar el problema a través de un pre-procesamiento para que cualquier hipercaja ubicado en el espacio pueda ser escalado y trasladado al origen y una vez terminadas las operaciones se devuelve al espacio donde estaba originalmente.

5. Paralelizar los procesos implementados para reducir el tiempo de ejecución.

Apéndice A

Graficas: Resultados de Intersección desde 3 hasta 20 dimenciones

En este apendice se incluyen todas las figuras de los experimentos presentados durante el capítulo 6, desde la tercera hasta vigesima dimensión. Se incluyen con el propósito de que el lector pueda corroborar las características de las que se hace mención en dichas secciones y puede ayudar al lector a realizar una mejor interpretacion de los resultados presentados.

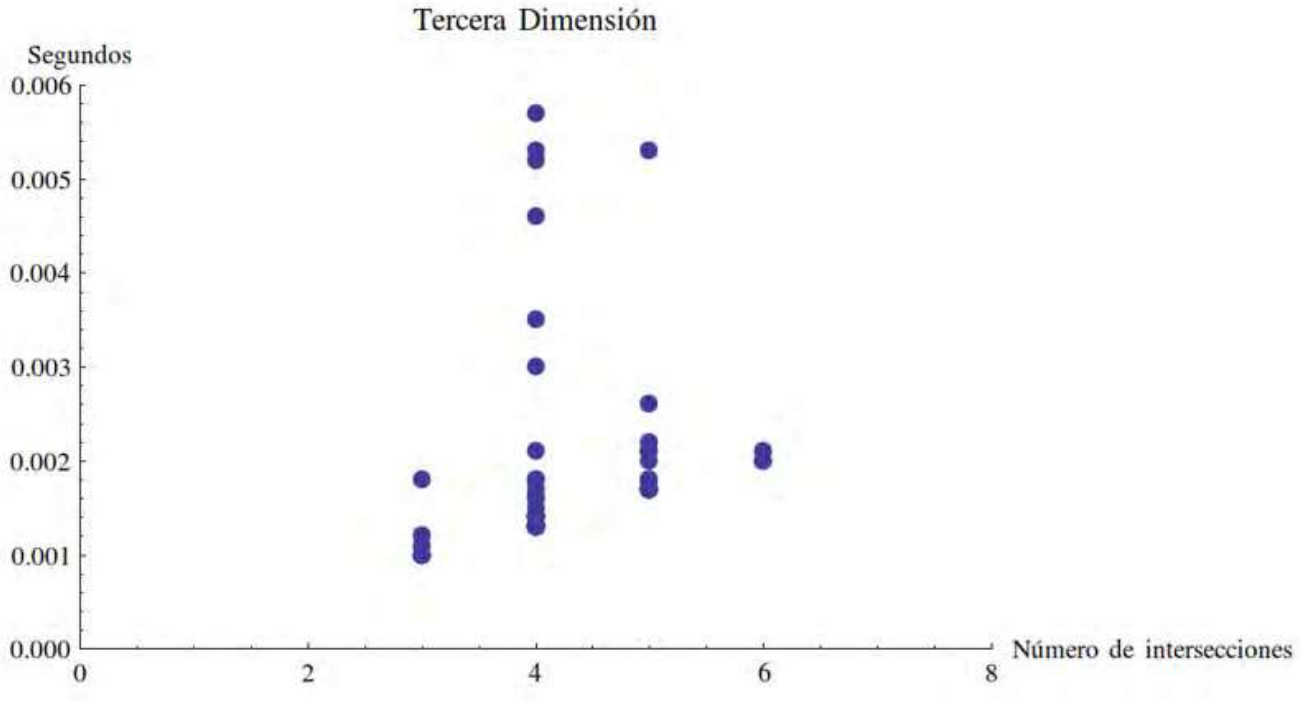


Figura A.1: Resultados obtenidos en la tercera dimensión.

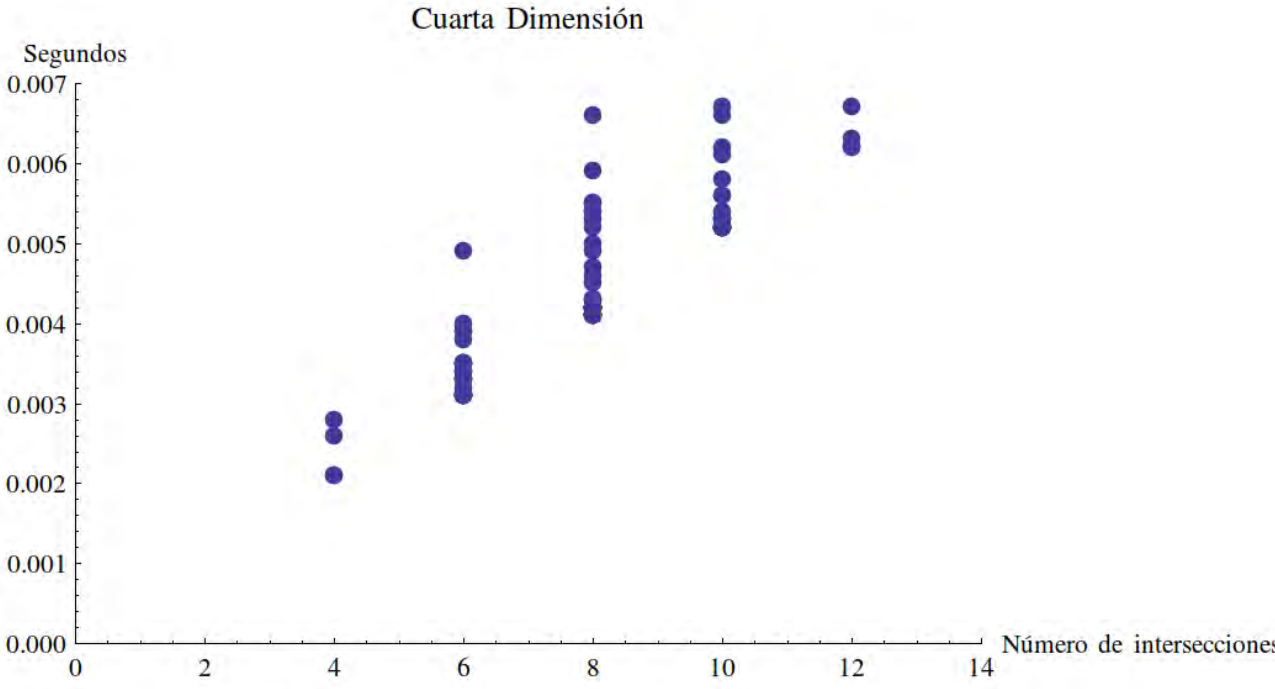


Figura A.2: Resultados obtenidos en la cuarta dimensión.

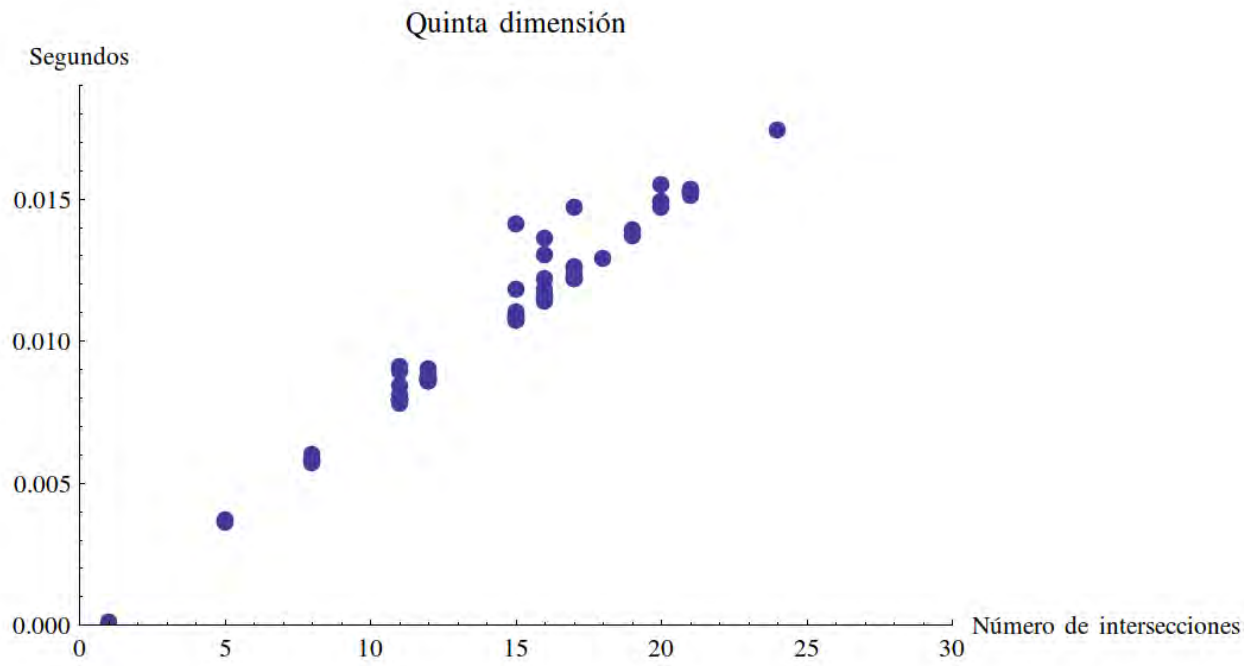


Figura A.3: Resultados obtenidos en la quinta dimensión.

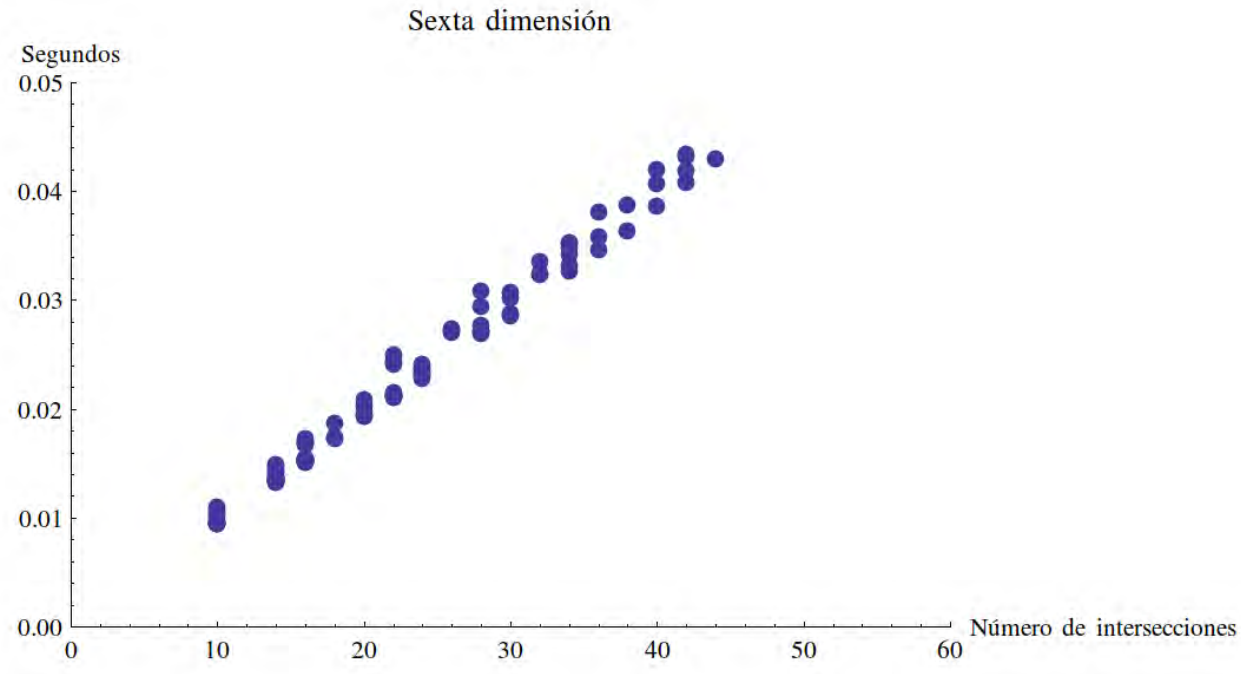


Figura A.4: Resultados obtenidos en la sexta dimensión.

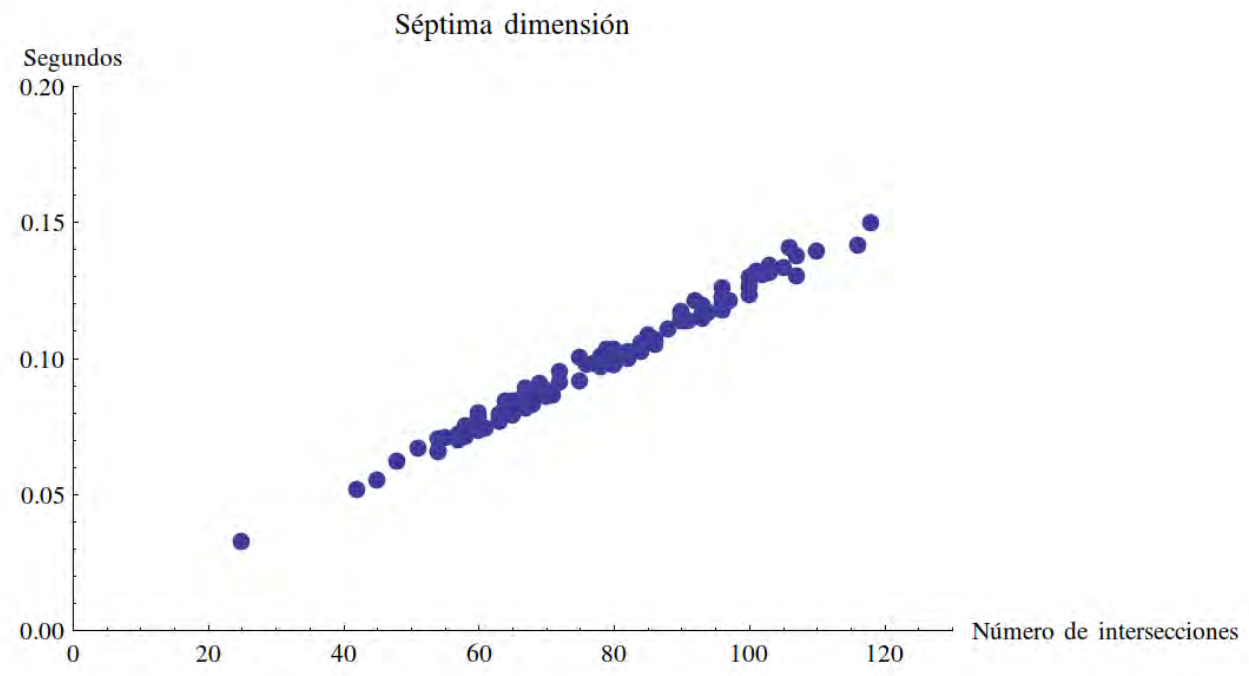


Figura A.5: Resultados obtenidos en la séptima dimensión.

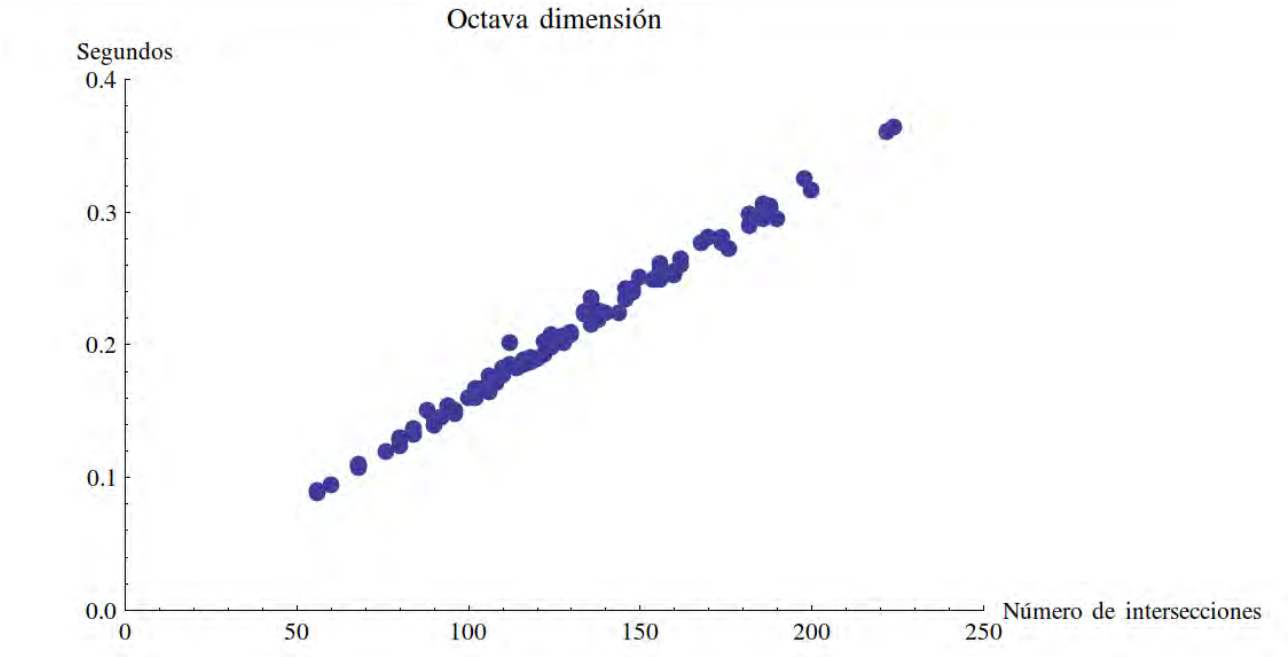


Figura A.6: Resultados obtenidos en la octava dimensión.

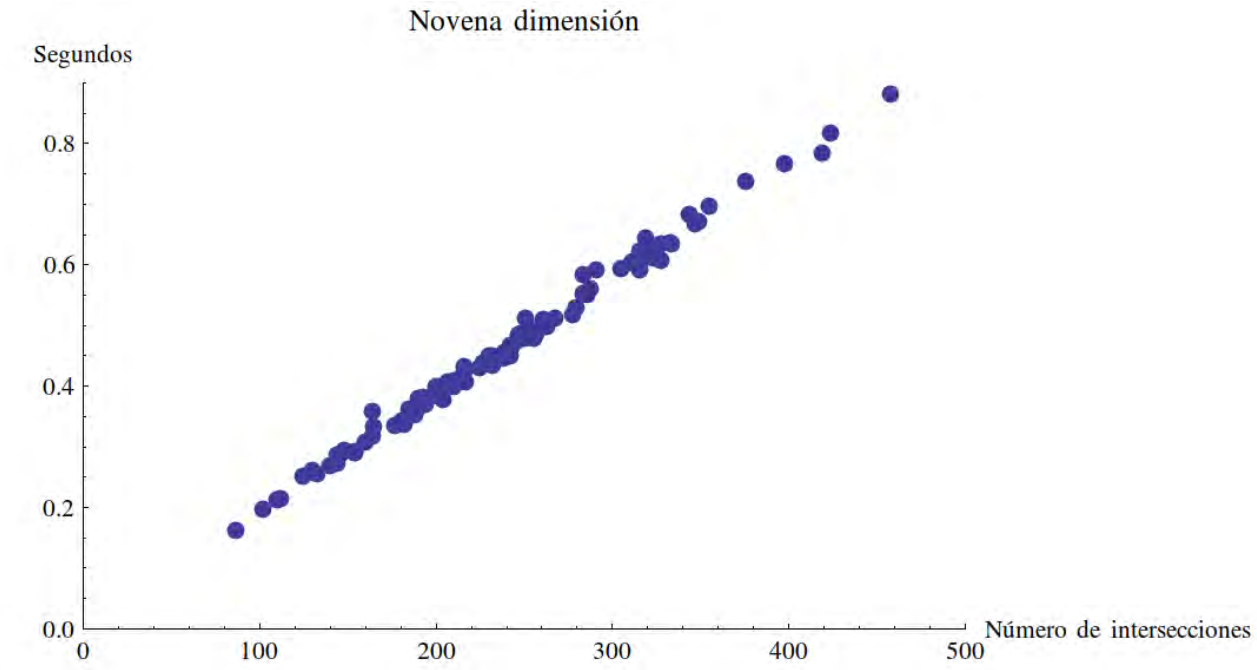


Figura A.7: Resultados obtenidos en la novena dimensión.

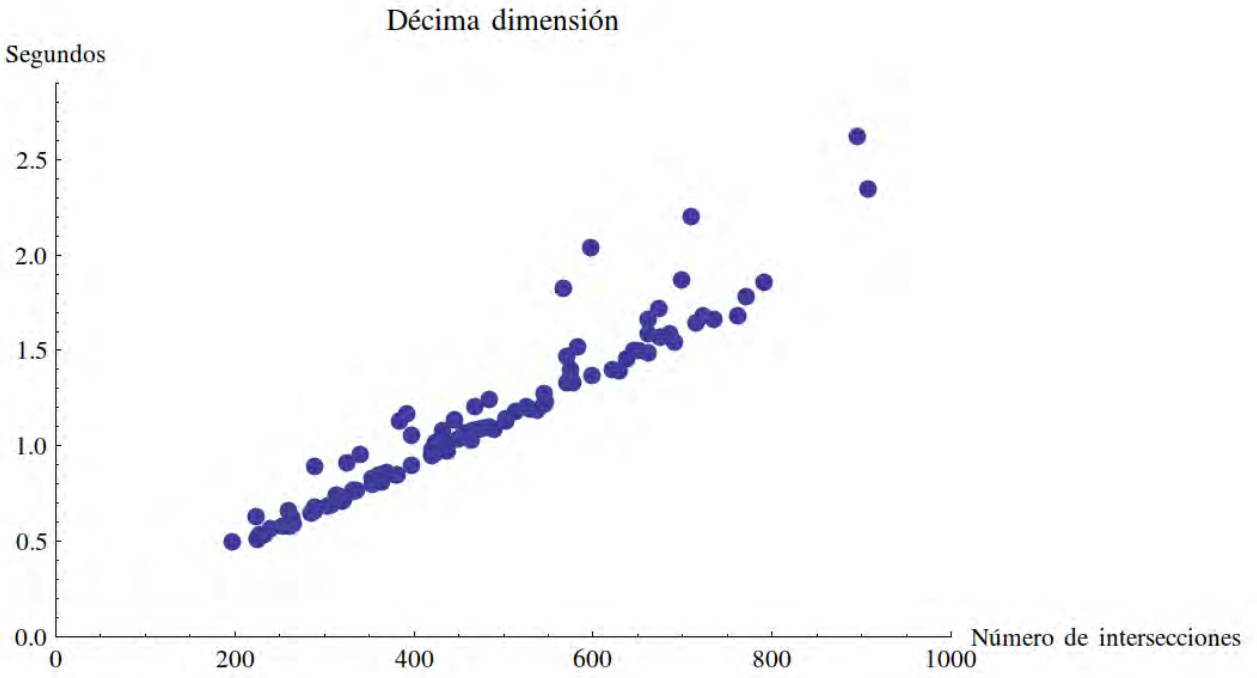


Figura A.8: Resultados obtenidos en la décima dimensión.

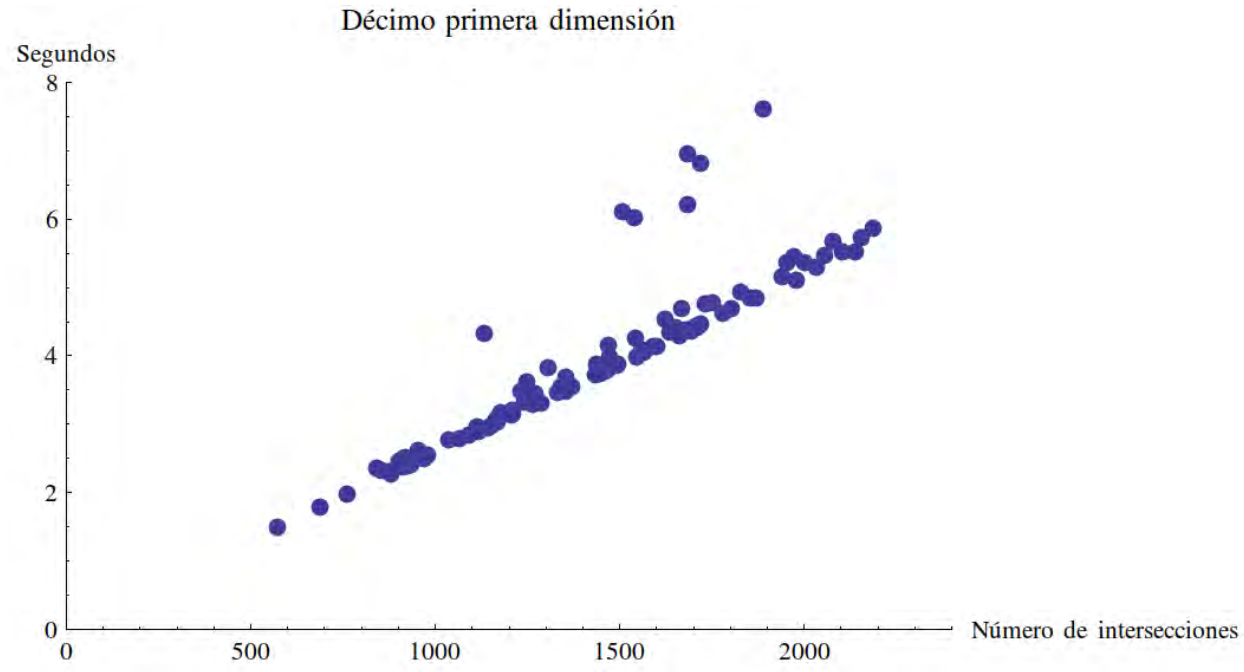


Figura A.9: Resultados obtenidos en la décimo primera dimensión.

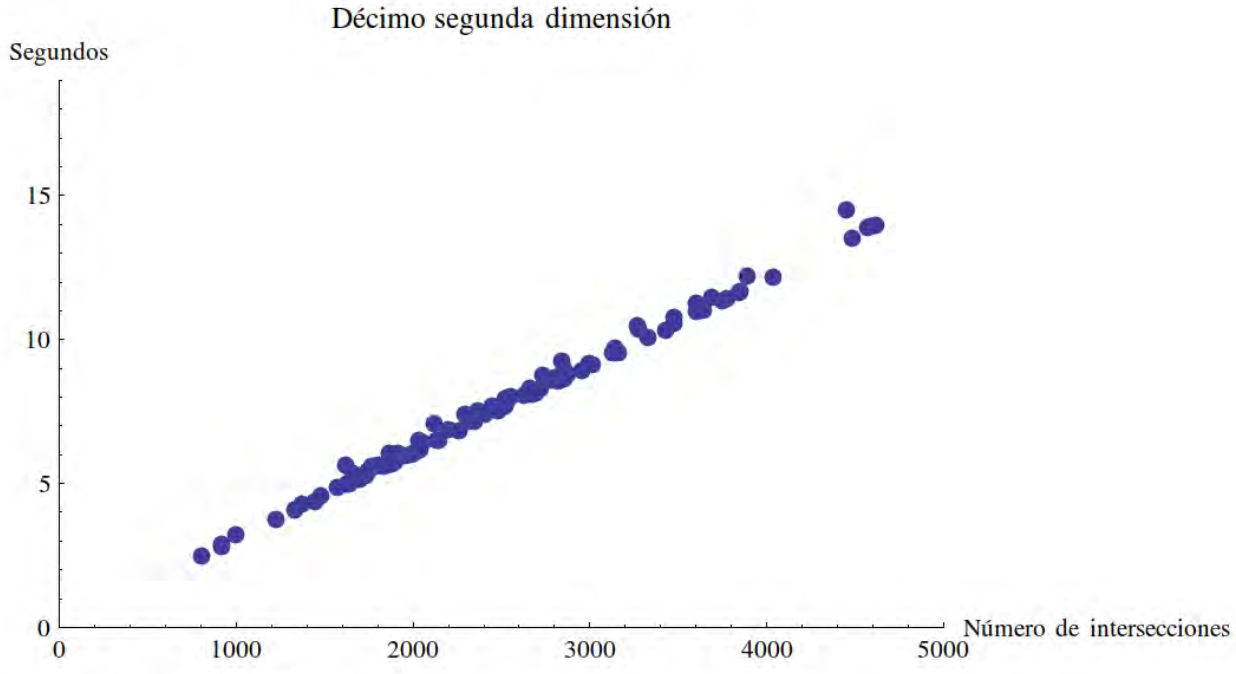


Figura A.10: Resultados obtenidos en la décimo segunda dimensión.

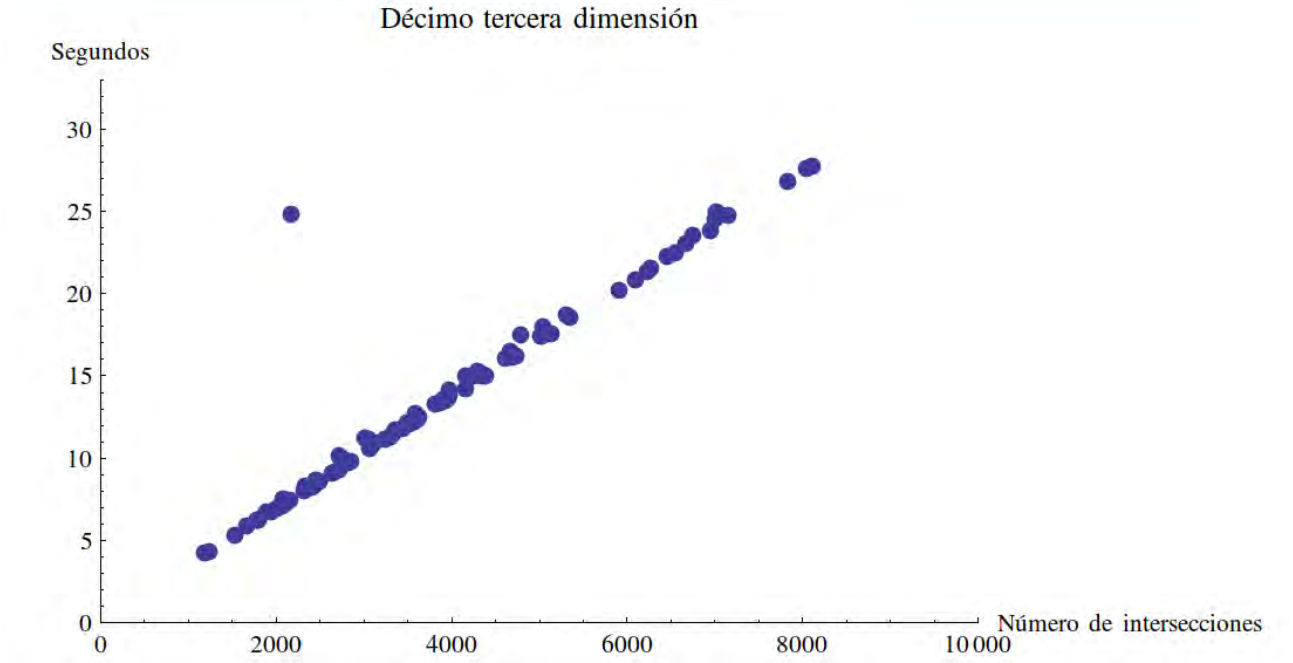


Figura A.11: Resultados obtenidos en la décimo tercera dimensión.

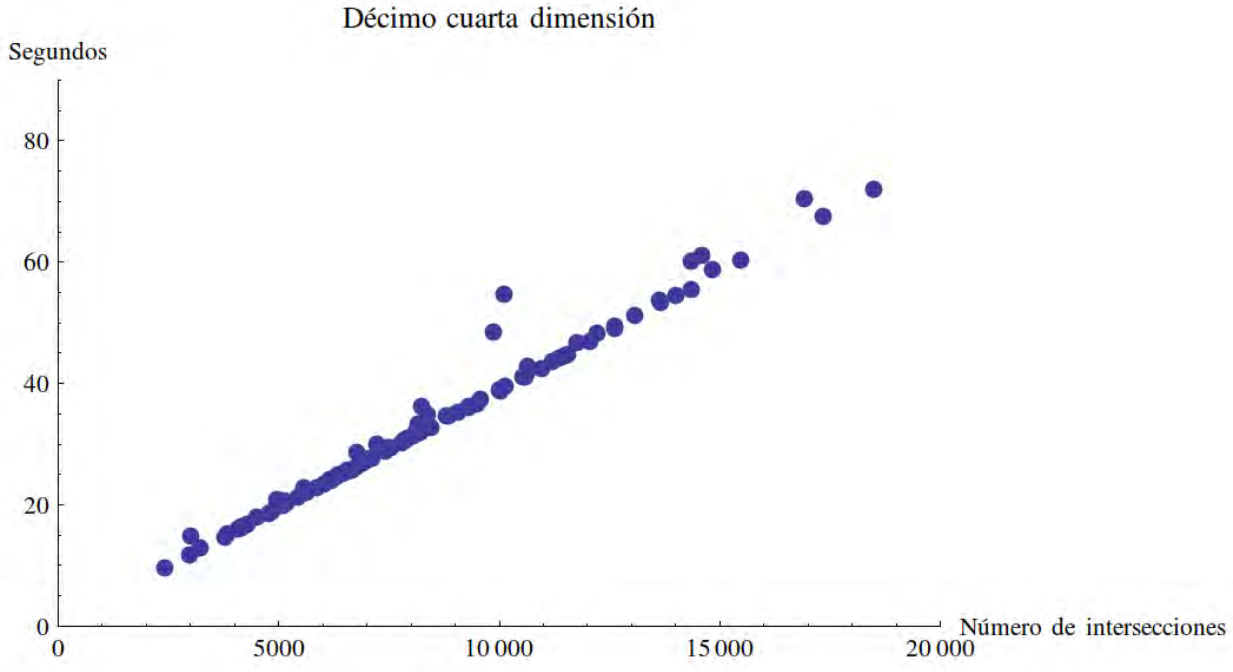


Figura A.12: Resultados obtenidos en la décimo cuarta dimensión.

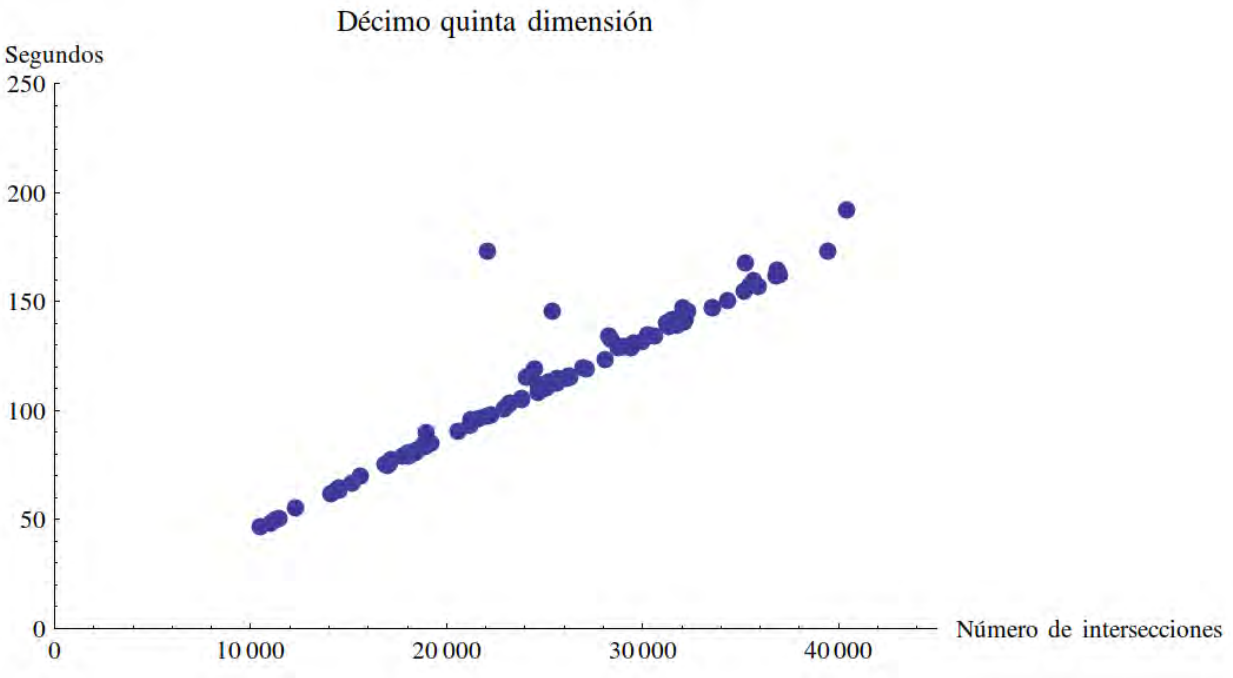


Figura A.13: Resultados obtenidos en la décimo quinta dimensión.

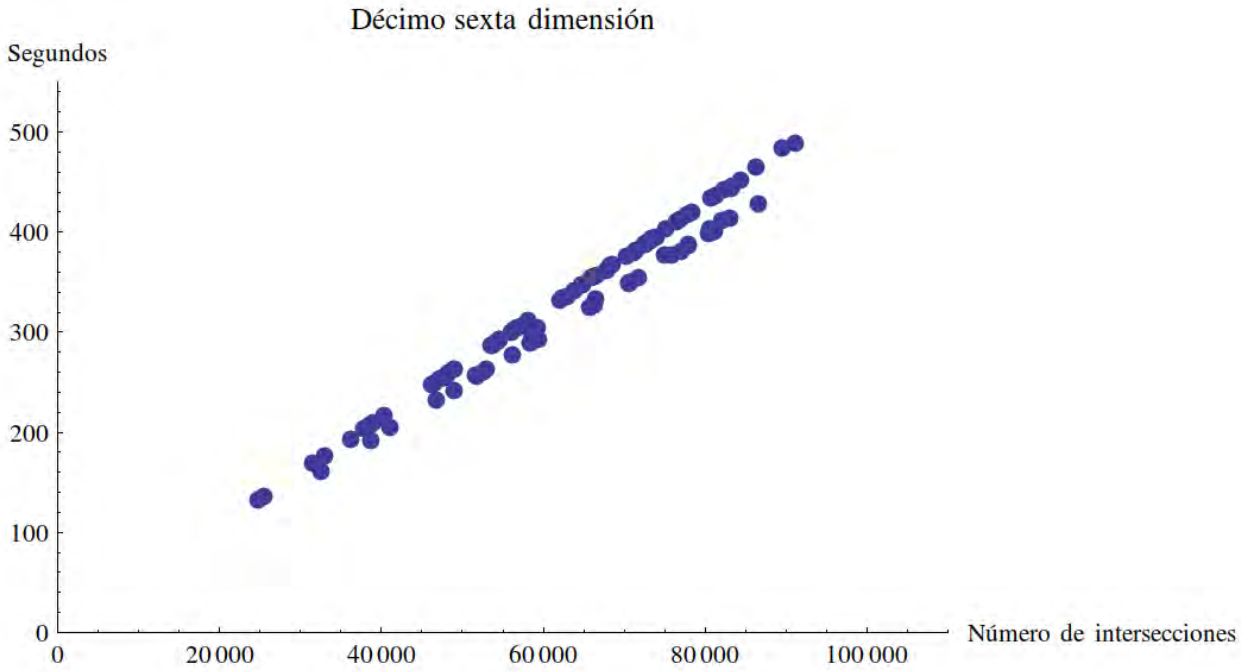


Figura A.14: Resultados obtenidos en la décimo sexta dimensión.

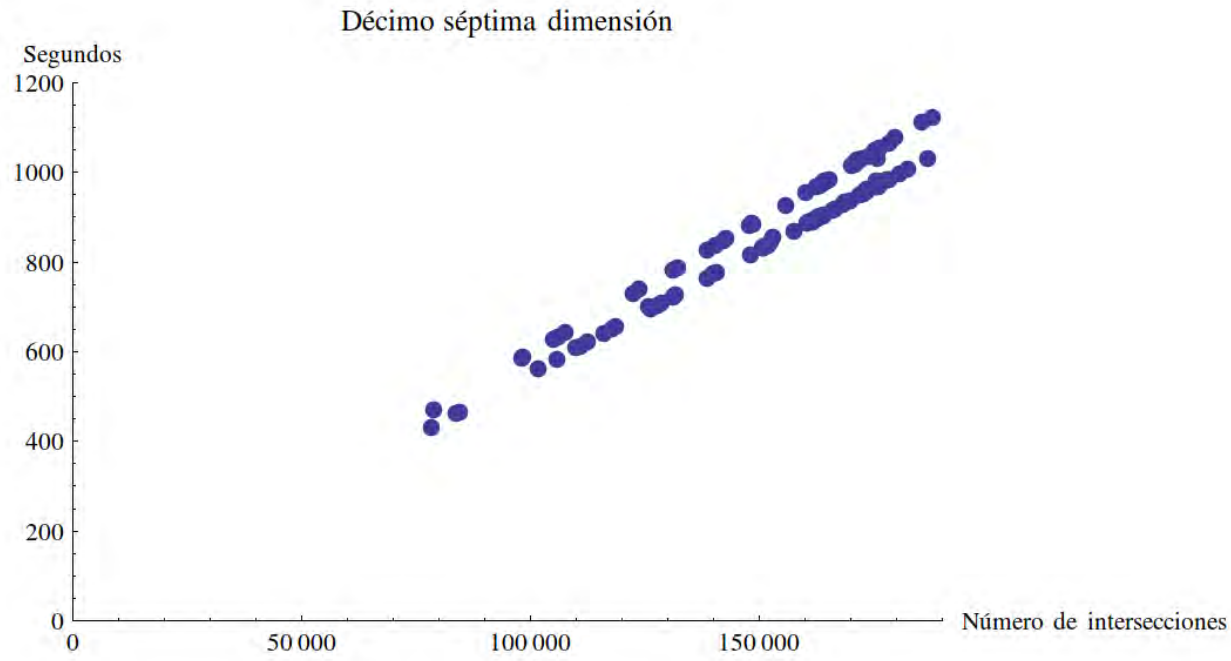


Figura A.15: Resultados obtenidos en la décimo séptima dimensión.

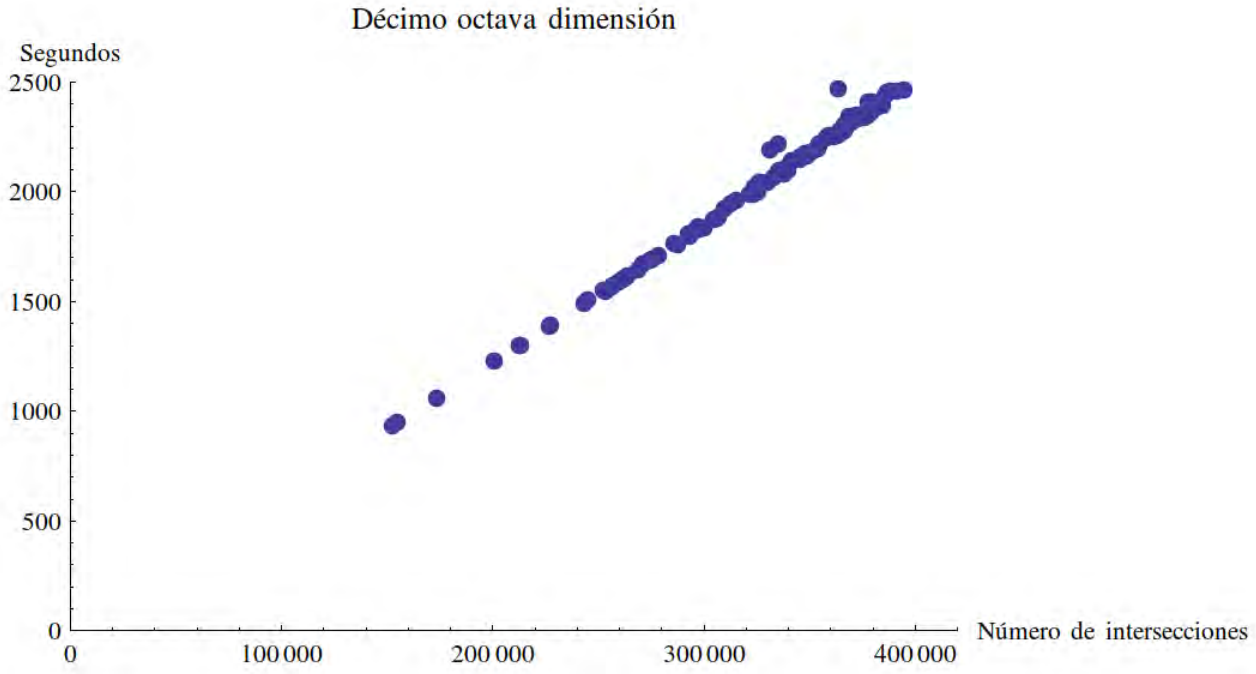


Figura A.16: Resultados obtenidos en la décimo octava dimensión.

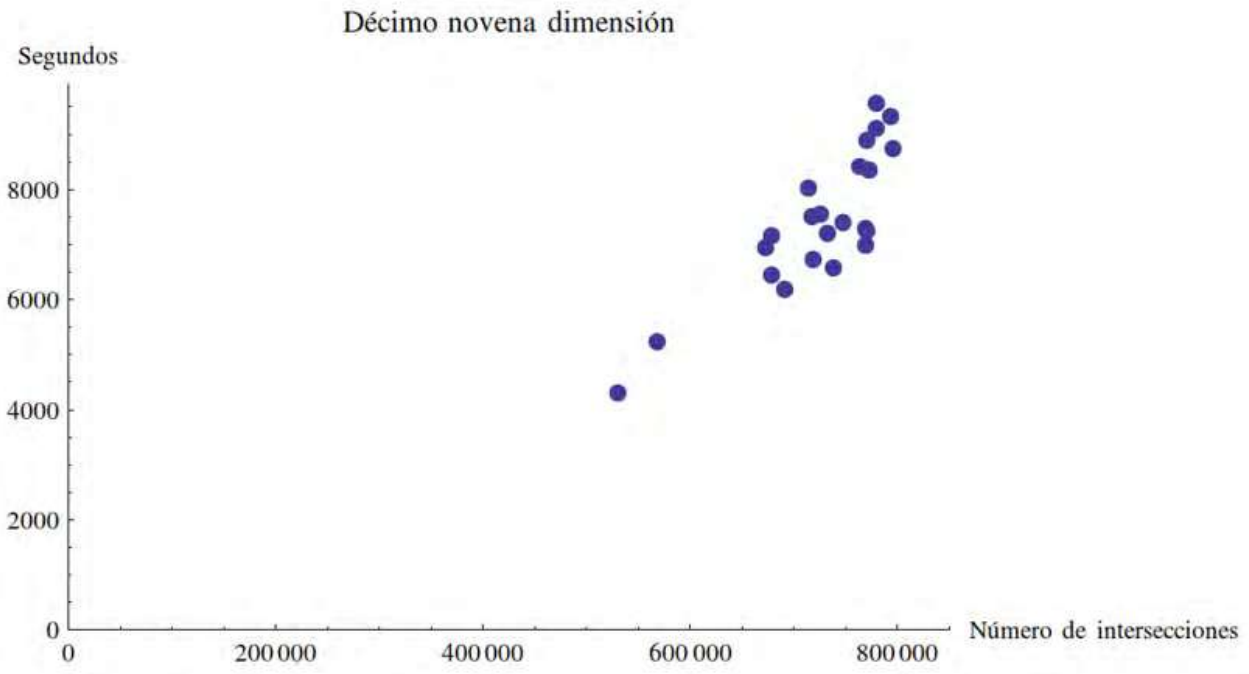


Figura A.17: Resultados obtenidos en la décimo novena dimensión.

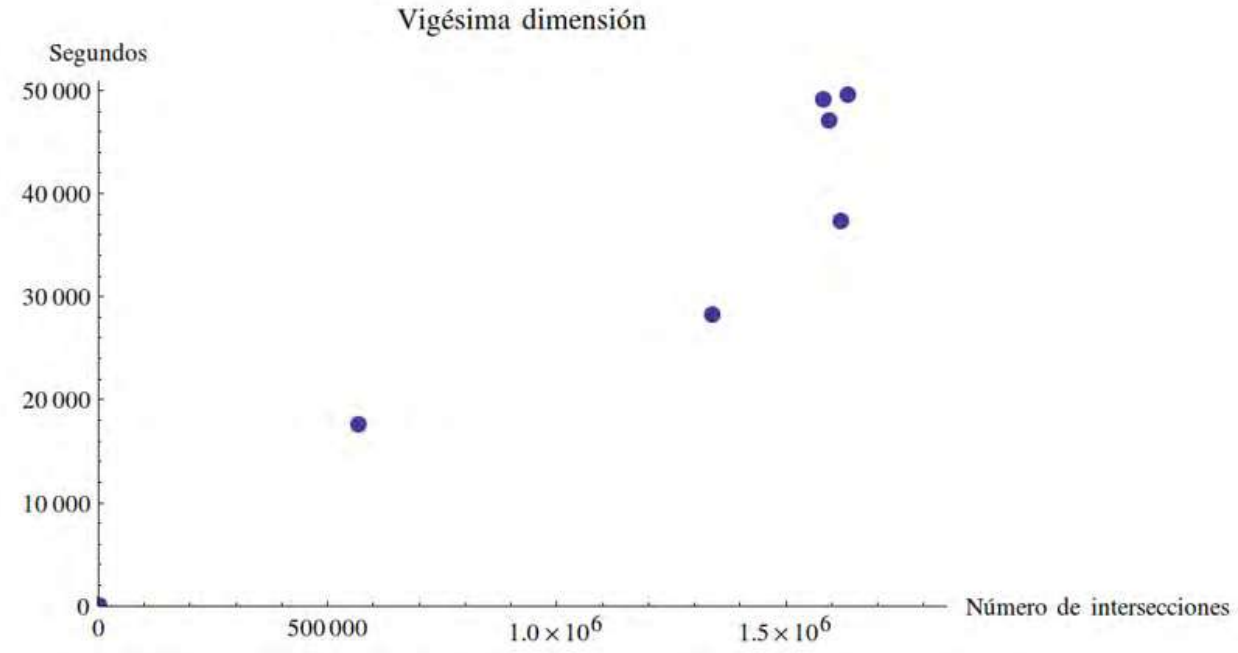


Figura A.18: Resultados obtenidos en la vigésima dimensión.

Referencias

[Ahlswede90] Ahlswede, R. y Zhang, Z. An identity in combinatorial extremal theory. *Advances in Mathematics*, 80(2):137–151, 1990.

[Bernard Kolman08] Bernard Kolman, D. R. H. *Algebra Lieal.* Pearson Education., 8ª ed$^{\underline{\text{ón}}}$., 2008.

[Calkin08] Calkin, N., James, K., Bowman, L., Myers, R., Riedl, E., y Thomas, V. Cuts of the hypercube. *Dept. of Mathematical Sciences, Clemson University (July 3, 2008)(manuscript)*, 2008.

[Chvatal83] Chvatal, V. *Linear programming.* Macmillan, 1983.

[Coto03] Coto, E. Algoritmos básicos de grafos. *Universidad Central de Venezuela Facultad de Ciencias Escuela de Computación*, págs. 1–26, 2003.

[Dantzig53] Dantzig, G. B. *Computational Algorithm of the Revised Simplex Method.* Report RM-1266, The RAND Corporation, Santa Monica, Calif,, 1953.

[Dantzig55] Dantzig, W. P., G. B. *The generalized simplex method for minimizing a linear form under linear inequality restraints.* Pacific Journal of Mathematics,, 1955.

[Dantzig63] Dantzig, G. B. *Linear Programming and Extensions.* Princeton University Press, 1963.

[Franco P. Preparata85] Franco P. Preparata, M. S. *Computational Geometry: An Introduction*. Springer, 1ª edón., 1985.

[García02] García, C., López, J. M., y Puigjaner, D. *Matemática discreta*. Pearson Educación, 2002.

[Gargano03] Gargano, M. L., Malerba, J. F., y Lewinter, M. Hypercubes and pascal's triangle: A tale of two proofs. *Mathematics magazine*, 76(3):216, 2003.

[G.Luenberger08] G.Luenberger, D. y Ye, Y. *Linear and Nonlinear Programming*. CA, USA: Standford University, 3ª edón., 2008.

[Graham72] Graham, R. L. y Pollak, H. O. On embedding graphs in squashed cubes. *En Graph theory and applications*, págs. 99–110. Springer, 1972.

[Grimaldi98] Grimaldi, R. P. *Matemáticas discreta y combinatoria: introducción y aplicaciones*. Pearson Educación, 1998.

[Grünbaum73] Grünbaum, B. Intersecting all edges of centrally symmetric polyhedra by planes. *Discrete Mathematics*, 5(3):241–246, 1973.

[Hernández01] Hernández, R., Lázaro, J. C., Dormido, R., y Torrecillas, S. R. *Estructuras de datos y algoritmos*. Prentice Hall, 2001.

[Julian07] Julian, H. *The practical revised simplex method*. School of Mathematics, University of Edinburgh January 25th 2007,, 2007.

[Klavžar] Klavžar, S. Counting hypercubes in hypercubes.

[Klee V72] Klee V, M. G. *How good is the Simplex algorithm?* Shisha O, editor. New York: Academic, 1ª edón., 1972.

[Lara09] Lara, C., Flores, J. J., y Calderon, F. On the hyperbox–hyperplane intersection problem. *INFOCOMP Journal of Computer Science*, 8(4):21–27, 2009.

[Luenberger89] Luenberger, D. G. y Mateos, M. L. *Programación lineal y no lineal.* 90C05 LUEp. Addison-Wesley Iberoamericana, 1989.

[Mount97] Mount, D. M. Geometric intersection. *En Handbook of Discrete and Computational Geometry, chapter 33.* Citeseer, 1997.

[O'neil71] O'neil, P. E. Hyperplane cuts of an n-cube. *Discrete Mathematics*, 1(2):193–195, 1971.

[Render06] Render, B., Stair, R. M., Hanna, M. E., et al. *Métodos cuantitativos para los negocios.* Pearson Educación, 2006.

[Rezk-Salama05] Rezk-Salama, . K. A., C. A vertex program for efficient box-plane intersection. *In Proc. Visions, Modeling and Visualization*, págs. 115–122, 2005.

[Shamos76] Shamos, M. I. y Hoey, D. Geometric intersection problems. *En Foundations of Computer Science, 1976., 17th Annual Symposium on*, págs. 208–215. IEEE, 1976.

[Sohler00] Sohler, C. y Ziegler, M. Computing cut numbers. 2000.

[Taha04] Taha, H. A. *Investigación de operaciones.* Pearson Educación., 2004.

[Vieira04] Vieira, H. y E.L., M. P. *An improved initial basis for the Simplex algorithm.* Production Engineering Program, COPPE-Federal University of Rio de Janeiro, Centro de Tecnologia, Bloco F, Sala F-105, Rio de Janeiro,, 2004. ISBN RJ 21945-970.

[Wagner.56] Wagner., H. M. *A comparison of the original and revised simplex method.* School of Industrial Management, Massachusetts Institute of Technology, Cambridge, Massachusetts,, 1956.

[Wirth80] Wirth, N., Rodríguez, A. A., y Bartolomé, J. C. *Algoritmos más estructuras de datos, programas.* Ediciones del Castillo, 1980.

[Witenberg00] Witenberg, J. P. *Métodos y modelos de investigación de operaciones*. Vol. 1 ed$^{\underline{\text{ón}}}$., 2000. ISBN Editorial Limusa.

Glosario de Términos

PL *Programación lineal.*

HHI *Hyperbox and Hyperplane Intersection method* - El método de Intersección enntre la Hipercaja e Hiperplano.

SBF *Solución basica factible.*

VLSI *Very Large Scale Integration.* - Integración a escala muy grande.

MSVNB Método simplex basado en columnas no básicas.

RHS *Right Hand Side.* - Lado derecho de la ecuación.

BFS *Breadth First Search.* - Búsqueda en anchura.

DFS *Depth First Search.* - Búsqueda en en profundidad.

LIFO *Last In First Out.* - Último en entrar primero en salir.