ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL

INSTITUTO DE CIENCIAS MATEMÁTICAS

ESCUELA DE GRADUADOS

PROYECTO DE GRADUACIÓN

PREVIO A LA OBTENCIÓN DEL TÍTULO DE:

"MAGÍSTER EN CONTROL DE OPERACIONES Y GESTIÓN LOGÍSTICA

TEMA

SOLUCIÓN AL PROBLEMA DE DISEÑO DE TERRITORIOS COMERCIALES Y RUTEO DE VEHÍCULOS

AUTOR:

ING. CARLOS SUÁREZ HERNÁNDEZ

Guayaquil- Ecuador

AÑO

2010

DEDICATORIA

Dedico
el presente trabajo
a mi familia y a mi novia,
por estar siempre
a mi lado
y apoyarme.

AGRADECIMIENTO

Agradezco a mi Director de Tesis por todo el apoyo proporcionado para realizar el presente trabajo.

DECLARACIÓN EXPRESA

La responsabilidad por los hechos y doctrinas expuestas en este Proyecto de Graduación, así como el Patrimonio Intelectual del mismo, corresponde exclusivamente al ICM (Instituto de Ciencias Matemáticas) de la Escuela Superior Politécnica del Litoral.

Carlos Suárez H.

Carlos Suárez Hernández

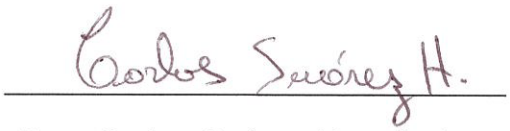
TRIBUNAL DE GRADUACIÓN

Ing. Pablo Álvarez
PRESIDENTE DEL TRIBUNAL

Phd. Luis Miguel Torres
DIRECTOR DE TESIS

Mg. Erwin Delgado
VOCAL DEL TRIBUNAL

AUTOR DEL PROYECTO



Ing. Carlos Suárez Hernández

TABLA DE CONTENIDO

ÍNDICE DE FIGURAS

Objetivo General

El objetivo general del proyecto consiste en proponer una solución al problema de diseño de territorios comerciales mediante la implementación de un modelo de programación matemática y de un método heurístico. Luego de lo cual, en cada territorio se empleará una heurística para determinar una secuencia eficiente de visita.

Objetivos Particulares

Para cumplir el objetivo general es necesario lograr los objetivos particulares que se presentan a continuación:

- Definir un modelo de programación lineal para el problema de diseño de territorios comerciales.
- Implementar el modelo de programación lineal de diseño de territorios en Scip.
- Seleccionar e implementar una heurística apropiada para el problema de diseño de territorios.
- Definir un modelo de programación lineal para el problema del agente viajero.
- Seleccionar e implementar una heurística para el problema del agente viajero.
- Analizar los resultados de los métodos exacto y heurístico para la solución del problema de diseño de territorios.

Introducción

La toma de decisiones fluye en toda organización desde los niveles estratégicos hasta los niveles operativos, y de acuerdo al nivel en que se tome la decisión, varía la complejidad de la misma y por tanto, su capacidad de producir los resultados deseados y los efectos secundarios más convenientes. Ante ello, surge la investigación de operaciones como una disciplina que proporciona poderosas herramientas, que permiten minimizar los márgenes de error y obtener el máximo provecho de los recursos disponibles.

Dado que en el sector de consumo masivo, el desempeño de la distribución comercial es una actividad que impacta directamente en la rentabilidad, la implementación de modelos matemáticos y métodos heurísticos para solucionar problemas de asignación de recursos constituyen un soporte importantísimo que no debe ser ignorado por los responsables de las actividades inherentes a la distribución comercial.

En el presente trabajo investigativo se plantea la aplicación de técnicas de optimización para contribuir al desempeño de la fuerza de ventas de una empresa de consumo masivo de la ciudad de Guayaquil. Se abarcarán dos problemas, uno a nivel táctico, como es el diseño de territorios comerciales, y otro operativo, el ruteo de vehículos. Para solucionar el problema de diseño de territorios comerciales balanceados respecto a diversos criterios, se implementarán dos alternativas: un modelo de programación lineal y un método heurístico. Posteriormente se compararán los resultados para varias instancias. En el caso del ruteo de vehículos, se implementará una heurística que determinará una secuencia eficiente de visita a los puntos de venta de cada territorio.

CAPÍTULO 1

1. DISTRIBUCIÓN COMERCIAL Y OPTIMIZACIÓN

La distribución comercial tiene como objetivo poner en contacto a los fabricantes o productores con los consumidores. Por lo que para que una empresa comercialice sus productos de forma efectiva, no son suficientes excelentes estándares de calidad en la fabricación, y que los productos sean conocidos y aceptados por el mercado, sino que además se encuentren disponibles en el lugar y momento adecuados. Para la consecución de tal objetivo intervienen factores intangibles como el nivel de gestión del vendedor en el punto de venta y su motivación, ésta última está definida en gran medida por el nivel de proporción entre su esfuerzo y sus ingresos.

El ambiente dentro de una fuerza de ventas es muy competitivo, y lo ideal es proporcionar un sistema de compensaciones o beneficios que contribuya a mantener un elevado nivel de motivación y compromiso organizacional entre sus miembros. Específicamente se tratará el caso de una empresa que distribuye de forma directa sus productos, para lo cual ha zonificado un área geográfica de interés en territorios comerciales, administrando cada uno como una unidad de negocio. El presente trabajo investigativo expone la aplicación de métodos de optimización para el diseño de territorios comerciales balanceados respecto a uno o varios criterios de interés, por ejemplo la carga de trabajo y beneficios económicos. Es decir que se indagará en el empleo de técnicas y modelos matemáticos para contribuir al desempeño de operaciones comerciales.

En base a lo expuesto observamos que la optimización de territorios comerciales contribuye tanto al clima laboral de la fuerza de ventas, como a los beneficios de la organización a la que pertenecen. Lo que finalmente se traducirá en un elevado nivel de servicio al mercado.

Posteriormente, debido a que cada territorio se considera como una unidad de negocio, es necesario organizar el trabajo de cada vendedor, de la mejor forma posible. Para lo cual, se propone el empleo de una técnica de ruteo de vehículos, con la finalidad de determinar secuencias óptimas de visita. Esto garantizará la disminución de tiempos improductivos en los territorios, permitiendo destinar mayores períodos a la atención de cada punto de venta, y no al desplazamiento entre los mismos.

CAPÍTULO 2

2. MARCO TEÓRICO

2.1. INTRODUCCIÓN A LA TEORÍA DE CONVEXIDAD

2.1.1. CONJUNTOS CONVEXOS

Los conjuntos convexos son los conjuntos más sencillos que aparecen de forma natural en la programación matemática. Su conocimiento es fundamental en la teoría de Optimización, a continuación se presentan algunas definiciones básicas, que serán de interés en la heurística de diseño de territorios implementada más adelante.

Conjunto Convexo.- Un conjunto C en E^n se dice convexo si para cada $x_1, x_2 \in C$ y cada número real α, $0 < \alpha < 1$, el punto $\alpha x_1 + (1-\alpha)x_2 \in C$.

Tal definición se puede interpretar geométricamente, afirmando que un conjunto es convexo si cada punto perteneciente a la línea recta que une dos puntos cualquiera perteneciente al conjunto convexo, también pertenecen al conjunto.

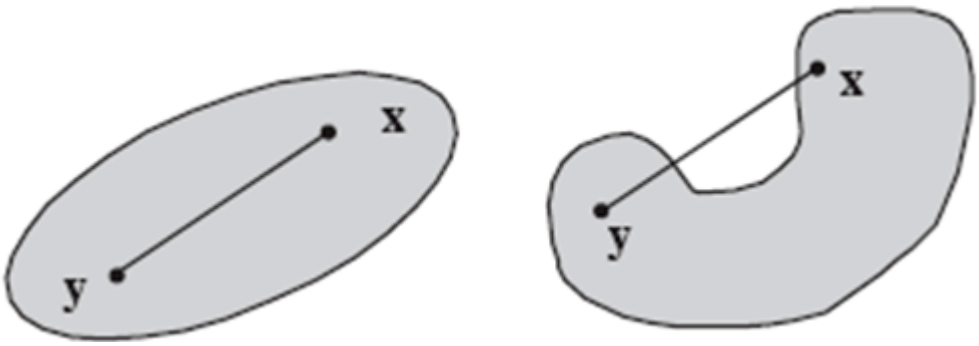


Fig 2.1.1.1 Conjunto Convexo y no Convexo

Las siguientes proposiciones muestran que ciertas operaciones entre conjuntos convexos conservan la convexidad.

Proposición

Los Conjuntos convexos en E^n, satisfacen las siguientes relaciones:

Si C es un conjunto convexo y β es un número real, el conjunto $\beta C = \{x : x = \beta c, c \in C\}$ es convexo.

Si C y D son conjuntos convexos, entonces el conjunto:

$C + D = \{x : x = c + d, c \in C, d \in D\}$ es convexo.

La intersección de cualquier colección de conjuntos convexos es convexa.

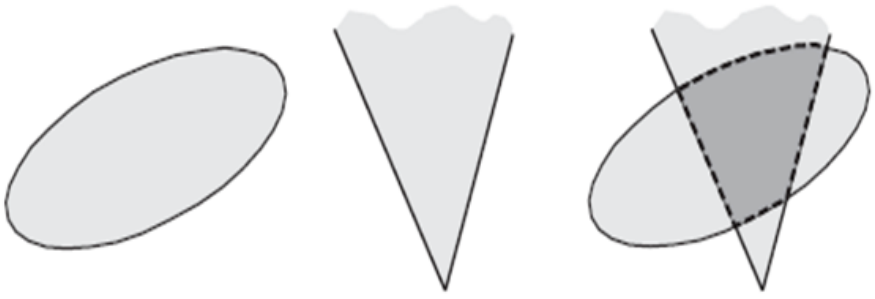


Fig 2.1.1.2 Intersección de conjuntos convexos.

2.1.2 CELDA CONVEXA

Sea S un subconjunto de E^n. La celda convexa de S, que se denotará por $co(S)$ es el conjunto que resulta de todas las intersecciones de todos los conjuntos convexos que contienen a S.

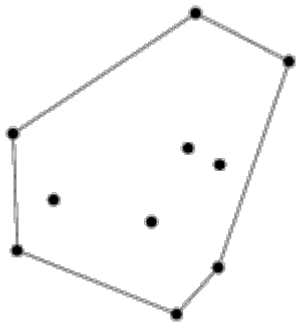


Fig 2.1.2.1 Celda Convexa

2.1.3 CONO

Un conjunto C es un cono si $x \in C$ implica que $\alpha x \in C$ para todo $\alpha > 0$. Un cono que es convexo, se conoce como cono convexo.

2.1.4 HIPERPLANOS Y POLITOPOS

En lo que respecta a la teoría de optimización, los hiperplanos son los conjuntos convexos más importantes. La definición más conocida de un hiperplano es la generalización de la definición geométrica de un plano en 3 dimensiones; pero desde el punto de vista de los algoritmos que permiten resolver problemas de programación lineal, es de mayor utilidad su definición algebraica. A continuación se presentarán varias definiciones y teoremas que describirán resultados de gran importancia, y que a su vez, más adelante serán relacionados con otros conceptos y resultados de

programación lineal. De esta manera se justificará la introducción a la teoría de convexidad en el presente estudio.

Variedad Lineal.- Un conjunto V en E^n, se denomina Variedad Lineal, si, dados X_1 y $X_2 \in V$, tenemos $\lambda X_1 + (1-\lambda)X_2$, para todo número real λ.

Cabe resaltar que la diferencia entre la definición de una variedad lineal, con la de un conjunto convexo, radica en el hecho de que en una variedad lineal la línea recta que pasa entre los dos puntos pertenece en su totalidad al conjunto, mientras que en un conjunto convexo, sólo el segmento de recta que une los dos puntos dados. En base a la definición se puede asociar una dimensión a una variedad lineal, por lo que un punto es una variedad lineal de dimensión cero, y una recta una de dimensión uno.

Hiperplano.- Un hiperplano en E^n es una variedad lineal de dimensión $(n-1)$..

Tal definición generaliza los conceptos de planos de dos dimensiones en un Espacio de tres dimensiones. En los siguientes enunciados se relacionará la definición algebraica con la geométrica.

Proposición.- Sea a un vector columna de dimensión n y c un número real. Entonces el conjunto:

$$H = \{X \in E^n : a^T X = c\}$$

Es un hiperplano en E^n.

Proposición.- Sea H un hiperplano en E^n. Entonces hay un vector no nulo de dimensión n y uno constante α tal que:

$$H = \{X \in E^n : a^T X = c\}$$

Combinando las dos proposiciones anteriores se observa que un hiperplano es el conjunto de soluciones para una ecuación lineal.

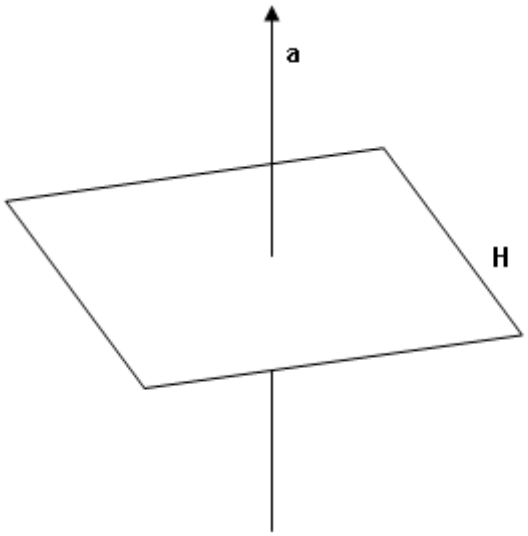


Fig 2.1.4.1 Hiperplano

En base a la definición de hiperplanos se tiene la siguiente definición:

Semiespacios Cerrados y Abiertos.- Sea α un vector no nulo en E^n y c un número real. Correspondientes al hiperplano

$$H = \{X \in E^n : aX = c\}$$

Entonces se definen como semiespacios cerrados de H a:

$$H_+ = \{X : a^T X \geq c\}$$

$$H_- = \{X : a^T X \leq c\}$$

Y semiespacios abiertos a:

$$H_{+}^{0} = \{X : a^T X > c\}$$

$$H_{-}^{0} = \{X : a^T X < c\}$$

Politopo Convexo.- Es un conjunto que puede ser expresado como la intersección de un finito número de semiespacios. Es decir que un politopo convexo es el conjunto solución obtenido de un conjunto de desigualdades de la forma:

$$a_1^T X \leq b_1$$

$$a_2^T X \leq b_2$$

.

.

$$a_m^T X \leq b_m$$

2.1.5 POLIEDRO

Un poliedro es un polítopo no vacío y cerrado.

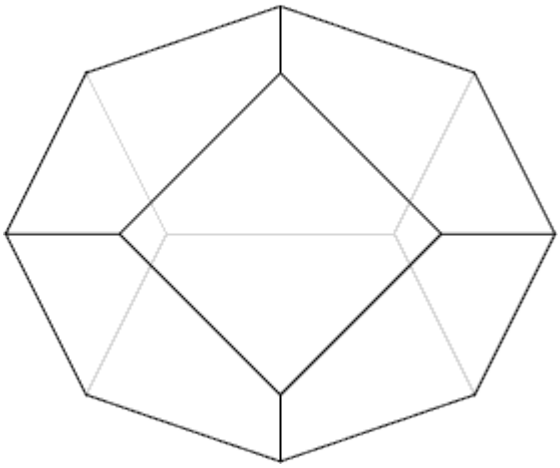


Fig 2.1.5.1 Poliedro.

Esta breve introducción a la teoría de convexidad se finalizará con la definición de Puntos extremos, un concepto muy importante para la programación lineal.

2.1.6 PUNTOS EXTREMOS

Un punto X en un conjunto convexo C es un punto extremo si no hay dos puntos diferentes X_1 y X_2 en C, tales que $X = \alpha X_1 + (1-\alpha)X_2$, para algún número real α, $0 < \alpha < 1$.

Las demostraciones de las proposiciones presentadas en esta sección se encuentran en **[1]**.

2.2. PROGRAMACIÓN LINEAL

La programación Lineal es una técnica de optimización, en la cual los problemas de interés se modelan mediante ecuaciones lineales. La posibilidad de modelar un gran número de procesos de diferentes disciplinas, la convierten en una poderosa y ampliamente utilizada técnica. Entre los problemas que pueden ser resueltos podemos mencionar:

- Asignación de horarios a empleados con el fin de mejorar la productividad y por otra parte elevar los niveles de satisfacción de los colaboradores de una organización.

- Selección de productos que se harán en etapas posteriores, aprovechando los recursos existentes y precios actuales para maximizar el rendimiento de los activos de una industria.

- Diseñar estrategias de distribución para empresas de consumo masivo, proponiendo tácticas que permitan minimizar los costos de distribución de productos a un mercado objetivo. Éste será el caso de estudio presentado más adelante en este trabajo investigativo.

En base a los ejemplos mencionados podemos afirmar que la programación lineal, básicamente proporciona modelos destinados a la asignación eficiente de recursos limitados en actividades a realizar, con el objetivo de satisfacer las metas propuestas, ya sea maximizando o minimizando el aprovechamiento de los recursos.

De forma general un problema de este tipo se puede representar de la siguiente manera:

Minimizar:

$$f = \sum_{j=1}^{n} c_j x_j$$

Sujeto a:

$$\sum_{j=1}^{n} a_{ij} x_j \le b_i, \quad i = 1,2,...,m$$

$$l_j \le x_j \le u_j, \quad j = 1,2,...,n$$

En el cual a_{ij}, b_i y c_j son matrices de coeficientes constantes, x_j las variables continuas. Finalmente l_j y u_j son los límites inferior y superior de las variables.

A continuación se presentan dos definiciones muy importantes relacionadas al espacio de soluciones factibles:

Solución Factible.- Un punto $X = (x_1, x_2, ..., x_n)$ que satisface todas las restricciones se denomina solución factible. El conjunto de todas estas soluciones se denomina región de factibilidad.

Solución Óptima.- Un punto factible Y tal que $f(X) \geq f(Y)$ para cualquier punto factible X, se denomina solución óptima del problema.

En los problemas lineales, existen ciertas propiedades que garantizan el óptimo global del mismo:

- Si la región factible está acotada, el problema siempre tiene una solución. Ésta es una condición suficiente pero no necesaria para que exista una solución.

- El óptimo de un problema de programación lineal es siempre un óptimo global.

- Si X y Y son soluciones óptimas de un problema de programación lineal, entonces cualquier combinación lineal convexa de los mismos también es una solución óptima.

2.2.1 PROBLEMAS LINEALES EXPRESADOS EN LA FORMA ESTÁNDAR

Luego de algunas manipulaciones todo problema se puede expresar de forma estándar:

Minimizar:

$$c_1 x_1 + c_2 x_2 + ... + c_{n1} x_n$$

Sujeto a:

$$a_{11}x_1 + a_{12}x_2 + ... + a_{1n}x_n = b_1$$
$$a_{21}x_1 + a_{22}x_2 + ... + a_{2n}x_n = b_2$$
$$.$$
$$.$$
$$.$$
$$a_{m1}x_1 + a_{m2}x_2 + ... + a_{mn}x_n = b_m$$

Con: $$x_1 \geq 0, x_2 \geq 0, ..., x_n \geq 0$$

Las manipulaciones algebraicas se resumen a continuación:

1. Las variables no restringidas en signo se expresan como diferencias de variables que sí están restringidas en signo, siendo no negativas. Si se tiene una variable no restringida en signo x_i, sus partes positiva y negativa se definen como $x_i^+ = \max\{0, x_i\}$ y $x_i^- = \max\{0, -x_i\}$.
2. Las restricciones de desigualdad se transforman en restricciones de igualdad mediante la introducción de variables de holgura.
3. Un problema de minimización es equivalente a uno de maximización, cambiando el signo de la función objetivo.
4. Una restricción con término independiente no positivo se puede reemplazar por otra equivalente cuyo término independiente es no negativo.

De una forma más compacta, el problema se expresa como:

Minimizar

$$C^T X$$

Sujeto a:

$$AX = b$$

Con $X \geq 0$

Donde X es un vector columna n-dimensional, C^T es un vector fila de dimensión n, A es una matriz mxn y b es un vector de dimensión m. La desigualdad $X \geq 0$ indica que cada componente es una variable no negativa.

2.2.2 SOLUCIONES BÁSICAS

Dado una matriz de la forma:

$$AX = b$$

Se asume que de las n columnas de A se seleccionan m columnas linealmente independientes, para facilitar la visualización tomaremos las m primeras columnas como linealmente independientes, y determinan una matriz mxn denominada B, que a su vez es no singular, por lo que no proporciona una solución única para la ecuación:

$$BX_B = b$$

Donde $X = (X_B, 0)$, en la cual los primeros m componentes de X, corresponden a los de X_B y los componentes restantes son iguales a cero. En base a lo cual se tiene la siguiente definición:

Soluciones Básicas.- Dado un conjunto de m ecuaciones simultáneas y n incógnitas. Sea B una matriz mxm no singular formada a partir de las columnas de A. Entonces los $n - m$ componentes de X no asociados a las columnas de B

son iguales a cero y la solución resultante del conjunto de ecuaciones se denominan solución básica con respecto a la

base B. Y los componentes de X asociados a columnas de B se nombran como variables básicas.

La matriz B es una base, por lo que está formada por m vectores linealmente independientes, y que generan el espacio E^m. La solución básica constituye una combinación lineal de los vectores base.

Es muy probable que el sistema de ecuaciones original $AX = b$ no tenga soluciones. Para lo cual se hacen ciertas suposiciones respecto de la estructura del mismo. En primer lugar se da como hecho que $n > m$, es decir que el número de elementos del vector X o variables exceden el número de restricciones y que las filas de A son linealmente independientes.

A continuación se presenta el teorema fundamental de la programación lineal, el mismo que establece la importancia de las soluciones básicas.

Teorema fundamental de la programación lineal.

Dado un problema de programación lineal en la forma estándar, donde A es una matriz de mxm de rango m, se tiene que:

- Si hay una solución factible, entonces es una solución básica factible.
- Si hay una solución óptima factible, entonces es una solución básica factible óptima.

2.2.3 RELACIONES CON CONVEXIDAD

Las conclusiones del teorema fundamental de programación lineal, tienen interpretaciones interesantes relacionadas con la teoría de Convexidad, que nos permiten entender de mejor forma las relaciones con la geometría. Principalmente la equivalencia entre soluciones básicas y puntos extremos, lo cual es formalizado en el siguiente teorema:

Teorema de Equivalencia de puntos extremos y soluciones básicas. Sea A una matriz mxm de rango m y b un vector de dimensión m. Sea K el polítopo convexo formado por todos los vectores de dimensión n, X que satisfacen la ecuación:

$$AX = b$$
$$X \geq 0$$

Un vector X es un punto extremo de K, sí solo sí, X es una solución básica factible del sistema $AX = b$.

Una demostración muy clara de los teoremas hasta aquí expuestos en esta sección está disponible en **[2]**.

Finalmente podría pensarse en un inicio en obtener el conjunto de todos los puntos extremos de un problema de programación lineal y determinar en cuál la función objetivo alcanza su valor mínimo; pero esto es una forma ineficiente de abordar tal solución, y costoso desde el punto de vista computacional.

El método más conocido para resolver problemas de programación lineal, el método simplex, es debido a Dantzig, quien lo introdujo en 1947. Afortunadamente, el crecimiento

de la capacidad de cálculo de los computadores ha permitido el uso de las técnicas desarrolladas en problemas de gran dimensión **[3]**.

2.2.4 PROGRAMACIÓN LINEAL ENTERA MIXTA

Muchos procesos representan problemas en los que además de variables continuas tales como flujos y costos, existen otras que permiten tomar decisiones como si se construyera o no alguna instalación, planificación de actividades, asignación de recursos etc. En tales casos se requiere que algunas o todas las soluciones factibles se puedan representar mediante números enteros. Si todas las variables son enteras, el problema se denomina de programación lineal entera. En el caso de que algunas variables son enteras y las demás son reales, se denomina un problema de programación lineal entera mixta. Finalmente nos podemos encontrar con casos en los cuales las variables indiquen si se lleva o no a cabo una actividad, por lo que tomarán dos valores posibles 0 o 1, este tipo de variables se denominan binarias, y es un caso especial de problemas de programación lineal entera.

La posibilidad de utilizar variables enteras o binarias a más de las reales, amplía el espectro de modelización matemática; pero también aumenta la complejidad de hallar soluciones factibles debido a los siguientes factores a considerar:

- En el caso de las variables enteras, lo expresado sobre teoría de convexidad y programación lineal para determinar los puntos extremos no se puede aplicar. Debido a que en términos generales los puntos extremos de la región factible no siempre serán enteros. Por lo que la solución

óptima se encontraría dentro de la solución factible en tal caso el método simplex, no proporcionaría la solución del mismo.

- Caso contrario con las variables reales, un problema de programación entera mixta, si tiene una región factible acotada, presentaría un número finito de soluciones. Por lo que una forma de resolverlos sería una exploración de todas las soluciones. Pero esto sería un proceso realmente ineficiente, debido a que en un problema con n variables binarias, habría que explorar 2^n.

Un problema de programación entera mixta, en su forma estándar, se formula de la siguiente forma:

Minimizar:

$$Z = \sum_{j=1}^{n} c_j x_j$$

Sujeto a:

$$\sum_{j=1}^{n} a_{ij} x_j = b_i, \quad i = 1,2,...,m$$

$$x_j \geq 0, \quad j = 1,2,...,n$$

$x_j \in N$, para todos o algunos $j = 1,2,...,n$

Donde $N = \{0,1,2,...\}$

Es un hecho que el algoritmo Simplex, por sí solo no es suficiente para resolver este tipo de problemas. Sin embargo existen diferentes algoritmos que abordan diferentes tipos de problemas, mejorando su rendimiento según la estructura del problema. El algoritmo más usado para resolver programas enteros y/o mixtos es el de ramificación y acotamiento **[4]**.

Este método resuelve una secuencia ordenada de problemas de programación lineal, que se obtienen relajando las restricciones de integralidad y añadiendo restricciones adicionales. El número de restricciones adicionales crece a medida que el procedimiento de ramificación progresa. Estas restricciones permiten separar la región factible en subregiones complementarias. El procedimiento ramificación y acotamiento establece inicialmente cotas inferior y superior del valor óptimo de la función objetivo. El mecanismo de ramificación aumenta progresivamente el valor de la cota inferior y disminuye también progresivamente el valor de la cota superior. La diferencia entre estas cotas es una medida de la proximidad de la solución actual a la óptima, si ésta existe. Al minimizar, se obtiene una cota inferior de la solución óptima relajando las restricciones de integralidad del problema inicial y resolviendo el problema de programación lineal resultante. De manera análoga, el valor de la función objetivo para cualquier solución del problema de programación entera mixta original es una cota superior de la solución óptima.

2.3. TEORÍA DE GRAFOS

El primer artículo científico relativo a grafos fue escrito por el matemático Suizo Leonhard Euler en 1736. Euler se basó en su artículo en el *problema de los puentes de Königsberg* . La ciudad de Kaliningrado, originalmente *Königsberg* , es famosa por sus siete puentes que unen ambas márgenes del río Pregel con dos de sus islas. Dos de los puentes unen la isla mayor con la margen oriental y otros dos con la margen occidental. La isla menor está conectada a cada margen por un puente y el séptimo puente une ambas islas. El problema planteaba lo siguiente: ¿es posible,

partiendo de un lugar arbitrario, regresar al lugar de partida cruzando cada puente una sola vez? **[5]**.

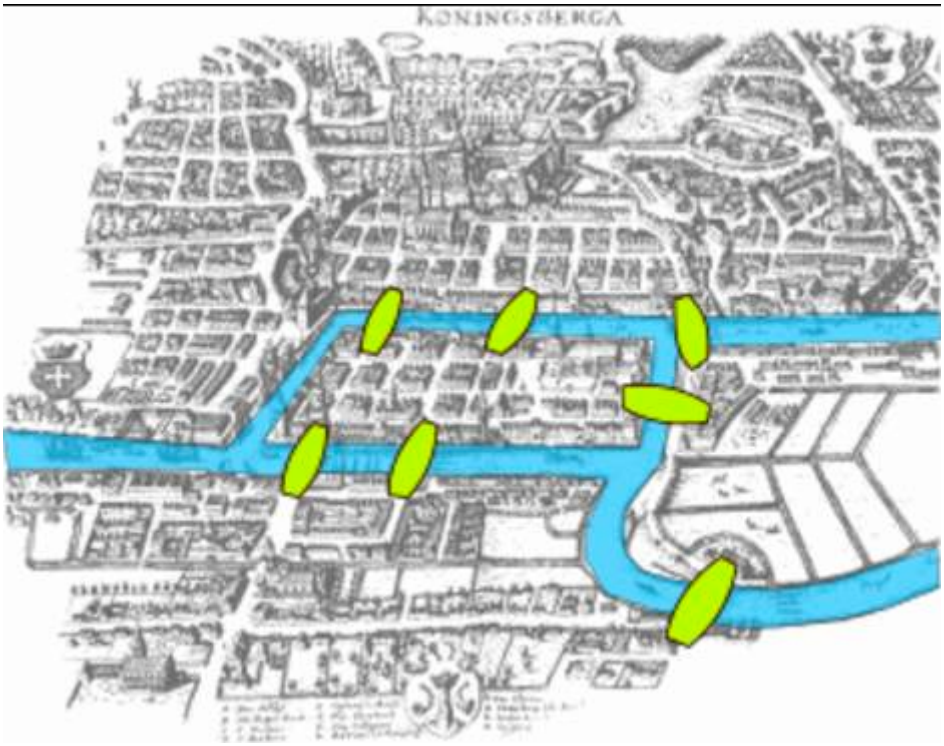


Fig 2.3.1 *Problema de los puentes de Königsberg*

Abstrayendo este problema y planteándolo con la (entonces aún básica) teoría de grafos, Euler consigue demostrar que el grafo asociado al esquema de puentes de Königsberg no tiene solución, es decir, no es posible regresar al vértice de partida sin pasar por alguna arista dos veces.

2.3.2 DEFINICIONES SOBRE GRAFOS

Grafo Dirigido.- Un grafo dirigido $G = (N, A)$ consiste de un conjunto de N nodos y un conjunto de arcos A de arcos cuyos elementos son pares ordenados de nodos diferentes.

Redes Dirigidas y no Dirigidas

Una red dirigida es un grafo dirigido cuyos nodos y/o arcos tienen asociados valores numéricos, por ejemplo: costos, capacidades, ofertas, demandas etc.

Una analogía de este concepto define a una red no dirigida.

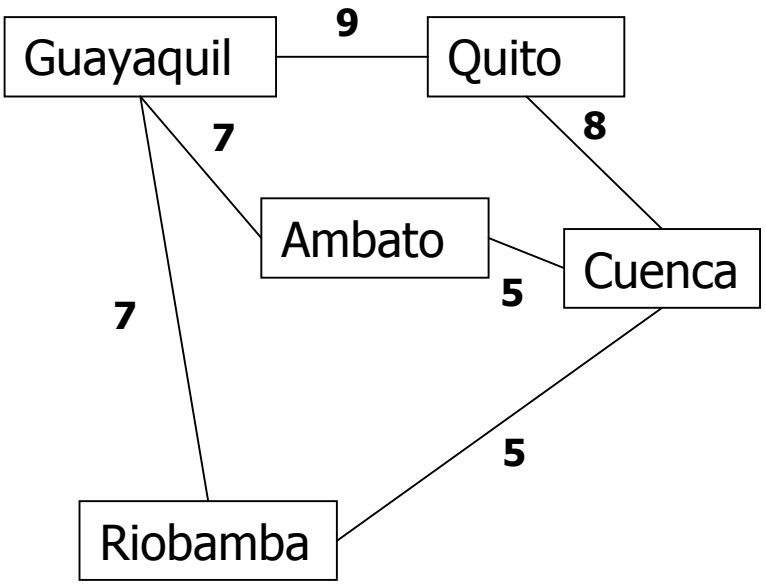


Fig 2.3.1.1 Grafo No Dirigido

Por ejemplo el sistema de carreteras en nuestro país se puede modelar mediante una red dirigida, donde cada arco tendría asociados valores correspondientes a: capacidad de flujo de vehículos, distancias, costos de transporte etc.

Grafo No Dirigido.- Un grafo no dirigido $G = (N, A)$ consiste de un conjunto de N nodos y un conjunto de arcos A de arcos cuyos elementos son pares no ordenados de nodos diferentes.

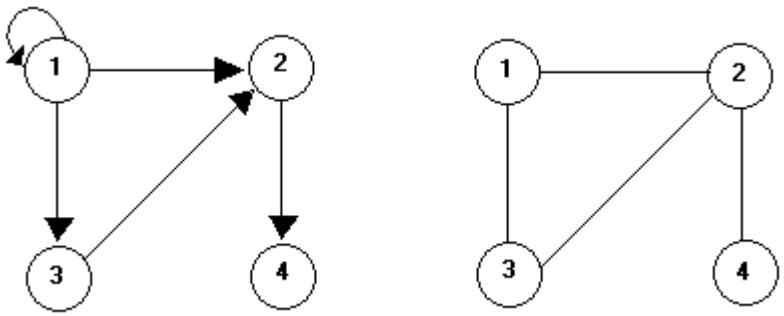


Fig 2.3.1.2. Grafo dirigido y Grafo no dirigido

En un grafo no dirigido un arco que une dos nodos i y j se puede expresar como (i,j) o (j,i). Por lo que un arco no dirigido puede ser considerado como una calle con doble sentido. De forma análoga un arco dirigido es considerado como una calle de una sola vía.

Colas y Cabezas.- Un arco dirigido (i,j) tiene dos puntos finales, i y j. El nodo i se denomina como cola, y el nodo j como cabeza. Se dice que el arco (i,j) emana del nodo i y termina en el nodo j. Un arco (i,j) es incidente a los nodos i y j. El arco (i,j) representa una salida de flujo del nodo i, y una entrada al nodo j. Si el arco $(i,j)\in A$, se dice que el nodo j es adyacente al nodo i.

Grados de un Nodo.- El grado entrante de un nodo, es el número de arcos que llegan al nodo. Grado saliente es el número de arcos o aristas que emanan de un nodo. Finalmente Grado de un nodo es la suma de los grados entrante y saliente.

Listas de Adyacencia.- La lista de arcos de adyacencia $A(i)$ de un nodo i es el conjunto de arcos salientes o que emanan del nodo, es decir $A(i)=\{(i,j)\in A: j\in N\}$. La lista de nodos de adyacencia $A(i)$, es el conjunto de nodos adyacentes al nodo, por lo que se definiría mediante $A(i)=\{j\in N:(i,j)\in A\}$.

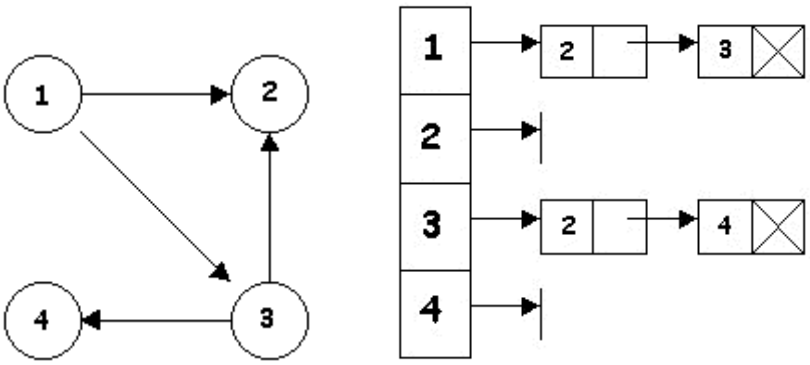


Fig.2.3.1.3 Grafo Dirigido y su Lista de adyacencia.

Multiarcos y Lazos.- Multiarcos son dos o más arcos que poseen los mismos nodos cabeza y cola. En cambio un lazo, es un arco en el cual la cabeza y cola, son el mismo nodo.

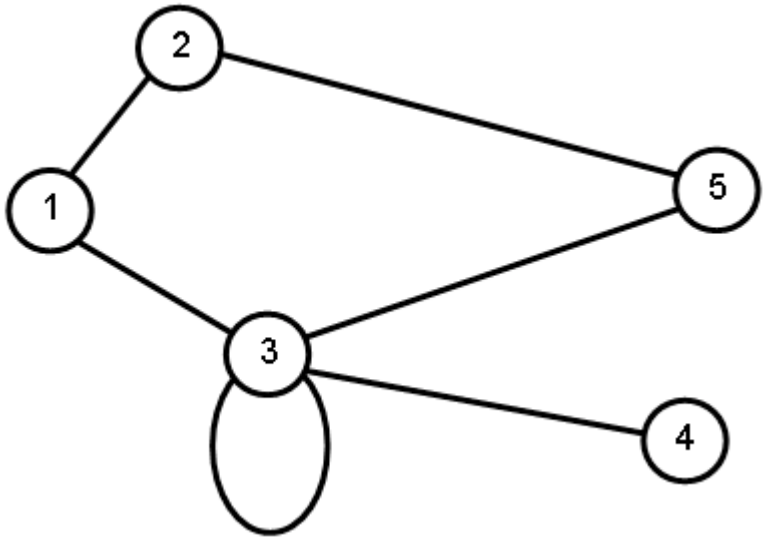


Fig.2.3.1.4 El nodo 3 pertenece a un lazo.

Una referencia completa de las definiciones de grafos y sus correspondientes representaciones computacionales están en **[6]**.

CAPÍTULO 3

3. EL PROBLEMA DE DISEÑO DE TERRITORIOS COMERCIALES

3.1. BALANCE DE TERRITORIOS COMERCIALES

La productividad de la fuerza de ventas es un aspecto muy importante en la gestión comercial de toda empresa de distribución de productos de consumo masivo. Constantemente se busca cómo mejorar los rendimientos del equipo de vendedores, con la menor inversión posible. Tal situación se justifica dado que la fuerza de ventas es una de las más caras inversiones en recursos humanos.

Las compañías llevan a cabo numerosas iniciativas que conducen al mejoramiento de la productividad de la fuerza de ventas. Las iniciativas de productividad de las fuerzas de ventas populares incluyen: automatización de las fuerzas de ventas, programas de administración, generación de sistemas de información, programas de telemarketing, iniciativas de formación, el uso de personal de medio tiempo o vendedores temporales.

Tales iniciativas son válidas; pero por lo general tienen un costo elevado y los rendimientos o crecimientos que se observan no son tan elevados. Ante ello el diseño de territorios comerciales balanceados en base a ciertos criterios definidos por la organización constituye una herramienta eficaz para obtener resultados notables en el corto plazo.

Es común en el mercado ecuatoriano encontrar empresas cuyos territorios comerciales están desbalanceados con respecto a

beneficios económicos y carga de trabajo. Por lo que demasiado esfuerzo es desplegado contra los clientes potenciales bajos y muy poco es desplegado en contra de muchos clientes potenciales fuertes.

3.1.1. BENEFICIOS DE UN ADECUADO DISEÑO DE TERRITORIOS COMERCIALES

Entre los beneficios tenemos los siguientes:

- Mejora en la distribución y participación de mercado.
- Incremento en ventas.
- Sistemas justos de recompensa.

3.1.1.1 MEJORA EN LA DISTRIBUCIÓN Y PARTICIPACIÓN DE MERCADO

Un vendedor en un territorio con carga excesiva de trabajo es incapaz de atender de forma efectiva a sus clientes, en la situación contraria, un vendedor en un territorio con muy poco trabajo gasta demasiado tiempo con clientes improductivos. En base a aquello es posible encontrar clientes insatisfechos pero potencialmente rentables en territorios muy extensos para ser atendidos por un solo vendedor, los mismos que pueden ser reasignados a vendedores que tienen capacidad disponible. El resultado es un incremento en la cobertura de clientes del mercado y por ende una mejora sustancial en la distribución.

3.1.1.2 CONTRIBUCIÓN DE UN DISEÑO EFICAZ DE TERRITORIOS AL NIVEL DE VENTAS

Cabe recalcar que las empresas de consumo masivo cuentan con un portafolio de marcas y productos, como elementos de su estrategia comercial para llegar a los diferentes segmentos de mercado. Un correcto diseño territorial le permite a una empresa aprovechar las oportunidades de mercados cada vez más dinámicos, compuestos por clientes en su mayoría selectivos y que valoran además de los aspectos monetarios, el nivel de servicio. Territorios compactos y balanceados respecto a la carga de trabajo, permiten a la fuerza de distribución llegar o atender de mejor forma a un conjunto de clientes, ofreciendo el mix adecuado de productos, garantizando el crecimiento vertical, el mismo que viene definido por la variedad de productos que se entrega a cada cliente.

3.1.1.3 SISTEMAS JUSTOS DE RECOMPENSA

Desde el punto de vista motivacional es muy importante que el equipo de vendedores disponga de un ambiente laboral que estimule y recompense económicamente de forma proporcional a la carga de trabajo. Por lo tanto los territorios deben estar diseñados de forma que la carga de trabajo esté balanceada, dentro de ciertas tolerancias. Para definir un sistema de recompensa la empresa debe tomar en cuenta los siguientes factores para el diseño territorial:

- La naturaleza básica del producto o servicio y su precio.
- Los compradores con quienes el vendedor habrá de interactuar incluyendo el número de compradores que son prospectos y el número que son clientes., quienes son, (agentes de compra, negocios al menudeo, o amas de casa, por ejemplo) donde están ubicados, una estimación de las compras anuales por cliente, y el número de compradores que un vendedor puede atender satisfactoriamente.
- El volumen de ventas necesario para poder apoyar al vendedor.
- El patrón de la interacción del vendedor incluyendo el número de interacciones hechas por día o semana y la frecuencia de las interacciones programadas para cada grupo de compradores.
- Identidad de los competidores.
- Facilidades para la transportación.
- La habilidad del vendedor.
- Tendencias en las ventas.

3.1.2 MODELO MATEMÁTICO PARA EL DISEÑO DE TERRITORIOS

El objetivo de resolver este problema, es lograr una agrupación óptima de los puntos de venta, o clientes de una empresa de consumo masivo, de forma que se diseñen territorios que contribuyan a maximizar la productividad de la

fuerza de ventas. Es común tener una cantidad considerable o elevada de puntos de venta que atender, éste es un factor que eleva la dificultad de los algoritmos de optimización para resolver este tipo de problemas. Con la finalidad de disminuir la complejidad de la región factible, se agrupan los puntos de venta, en unidades básicas que serán tomadas en cuenta como un solo ente a asignar a un territorio. Las restricciones del modelo deberán estar definidas en base a criterios geográficos, y económicos correctamente definidos, siempre con la visión de tener una adecuada administración de los puntos de venta y garantizar el efectivo abastecimiento del mercado.

El considerar a cada manzana como una unidad básica, implica que sus ventas y carga de trabajo corresponden a la suma de las actividades mencionadas de los clientes ubicados en dicha manzana. Es de interés que los territorios estén balanceados respecto a la carga de trabajo y beneficios económicos para la fuerza de ventas. Adicionalmente desde cualquier unidad básica se debe poder llegar a cualquier otra unidad básica dentro del mismo territorio. Tal característica nos permitirá, como se presentará en el siguiente capítulo de la tesis, aplicar métodos de optimización de ruteo para viajar entre unidades del mismo territorio sin abandonar el mismo.

Los territorios deben ser lo más compactos posible, es decir que las unidades básicas presentes en cada territorio deben estar cercas entre sí en la mayor medida posible. En resumen se consideran los siguientes criterios de planeación:

- Cada unidad básica debe estar asignada a un solo territorio.
- Los territorios deben ser compactos.

- Los territorios deben estar balanceados respecto a cada medida de actividad (carga de trabajo y beneficios para la fuerza de ventas).
- El número de territorios debe ser un parámetro del modelo.

La red de distribución comercial de la empresa se representa mediante un grafo $G=(V,E)$, y cada nodo $i \in V$ constituye una unidad básica, y una arista o arco $(i,j) \in E$ existe, si los nodos o en este caso manzanas i,j son adyacentes. Cada unidad básica $i \in V$ tiene asociada una coordenada Geográfica (C_i^x, C_i^x) y dos medidas de actividad. Se denotará por w_i^a, donde $a=1$ representa la carga de trabajo y $a=2$ los beneficios de los vendedores. Tal conjunto se denominará $A=\{1,2\}$.

Minimizar

$$f(x)=\sum_{i,j \in V} d_{ij}x_{ij}$$

Sujeto a:

$$\sum_{i \in V} x_{ij}=1 \qquad j \in V \quad (1)$$

$$\sum_{i \in V} x_{ii}=p \quad (2)$$

$$\sum_{j \in V} w_j^a x_{ij} \le (1+\tau)\mu^a x_{ii} \quad i \in V,\ a \in A \quad (3)$$

$$\sum_{j \in V} w_j^a x_{ij} \le (1-\tau)\mu^a x_{ii} \quad i \in V,\ a \in A \quad (4)$$

$$x_{ij} \in \{0,1\} \qquad i,j \in V \quad (5)$$

En el modelo anterior, se ha definido como función objetivo a la suma de las distancias entre las unidades básicas, esto sirve como una medida de dispersión, y su minimización

nos conducirá a soluciones relativamente compactas. El primer conjunto de restricciones (1), garantiza que cada unidad básica pertencerá a un solo territorio. En tanto que (2) establece que se diseñarán p territorios. (3) y (4) se refieren a los criterios de balance, y las tolerancias para las medidas de los criterios, que deben cumplir los territorios para ser considerados factibles. Tales tolerancias se refieren a desviaciones del promedio de cada actividad, su vez definido por $\mu^a = \frac{\sum_{i \in V} w_i^a}{p}$ (5) indica la naturaleza binaria de las variables de decisión. El modelo citado fue tomado de un estudio computacional del problema de diseño de territorios comerciales de una embotelladora de México **[7].**

Cabe mencionar que no se incluyen las restricciones de contigüidad territorial en la implementación, debido a que su número aumenta exponencialmente en función del número de unidades básicas del problema. Sin embargo, dicho conjunto de restricciones de contigüidad viene dado por:

$$\sum_{j \in \bigcup_{v \in S} N^v / S} x_{ij} - \sum_{j \in S} x_{ij} \geq 1 - |S| \qquad i \in V, S \subset V / \left(N^i \cup \{i\}\right)$$

Estas restricciones garantizan la conectividad de los territorios, donde N^i representa el conjunto de nodos adyacentes al nodo i **[8].**

3.1.3 HEURÍSTICA GEOMÉTRICA

Básicamente la heurística consiste en particionar el problema geométricamente mediante líneas rectas en pequeños subproblemas, hasta un nivel en el cual el problema original de diseño de territorios esté resuelto.

En primer lugar definimos a V como el conjunto de unidades básicas, que conformarán los territorios y cada territorio es un subconjunto $B \in V$. El objetivo del método es diseñar territorios balanceados respecto a una actividad medible, como por ejemplo el beneficio económico de los vendedores responsables de cada uno, o en base a la carga de trabajo que demanda cada territorio. Se define además a $W(B)$ como la medida de la actividad a balancear, correspondiente al territorio B, y p es el número de territorios que se pretende balancear. Por lo tanto el caso ideal sería contar con un diseño tal que $W(B) = \frac{W(V)}{p}$, pero tal situación es imposible, dado el tipo de problema. Debido a esto se definen tolerancias para las medidas de las actividades en cada territorio, dadas por L y U, donde L es el límite inferior y U el superior. Por lo que la medida de cada uno de los territorios que sean parte de la solución deben pertenecer a este rango para garantizar la factibilidad de la misma. Una forma de calcular estos límites podría ser en base a una tolerancia τ respecto al caso ideal, de tal forma que:

$$L = (1-\tau)\frac{W(V)}{p} \quad \text{y} \quad U = (1+\tau)\frac{W(V)}{p}$$

Un territorio B se denomina factible si $L \leq W(B) \leq U$.

La operación básica de la heurística consiste en dividir un subconjunto $V' \subseteq V$ de unidades básicas en dos mitades V_i' y V_d' mediante una línea recta sobre el plano donde están estos puntos. V_i' y V_d' son los subconjuntos disjuntos de unidades básicas ubicados a la izquierda y a la derecha de la línea recta respectivamente. Por lo que cada problema genera dos subproblemas que serán almacenados en un árbol binario.

La heurística explora el árbol binario donde cada nodo corresponde a un problema y finaliza cuando todas las hojas son generadas.

El problema inicial es particionar V en p territorios. Cada instancia de un problema básico está definida por $V' \subseteq V$ y un número positivo $p' \leq p$. De forma que un problema $p' = 1$ ya está resuelto porque V' representa un solo territorio. Si $p' > 1$ se resuelve seleccionando una línea recta y dos números positivos tales que $p_i', p_d' \geq 1$ con $p_i' + p_d' = p'$, lo cual origina dos subproblemas (V_i', p_i') y (V_d', p_d') que reemplazan al problema (V', p').

Al resolver cada problema se debe tener en cuenta los criterios de compacidad y balance respecto a la medida de una actividad. La forma en que p_i' y p_d' son seleccionados depende si p' es par o impar. Si p' es par se tiene que $p_i' = p_d' = p'/2$. Si es impar se consideran los siguientes dos casos.

$$p_i' = \left\lceil \frac{p'}{2} \right\rceil,\ p_d' = \left\lfloor \frac{p'}{2} \right\rfloor \text{ y } p_i' = \left\lfloor \frac{p'}{2} \right\rfloor,\ p_d' = \left\lceil \frac{p'}{2} \right\rceil$$

El particionar cada conjunto mediante una línea recta tiene dos ventajas explicadas mediante los siguientes enunciados:

Proposición 1: Si el problema básico es resuelto mediante la partición de V' en dos mitades por una línea recta, obtendremos dos territorios los cuales definirán dos celdas convexas disjuntas.

Proposición 2: Sino se da que tres puntos o unidades básicas de V' pertenezcan a una línea común, el número de particiones de V' a lo largo de la línea está limitada por $|V'|^2$.

3.1.3.1 GENERACIÓN DE PARTICIONES

La proposición 2 limita el número de particiones, lo cual sigue siendo un límite muy grande para problemas de gran magnitud. Por lo que no se examinan todas las particiones sino las generadas en ciertas direcciones. Si se escoge una dirección dada por $\alpha \in [0, \pi)$, sea $\alpha = \frac{1}{2}$ el ángulo de la línea con el eje positivo "x" por lo que se considerarán líneas paralelas al eje "y". Previo a examinar las particiones generadas se ordenan los puntos o unidades básicas en V' en orden no decreciente según el eje de coordenadas "x", cada posible partición a lo largo de una línea paralela al eje "y" divide a la secuencia ordenada en dos partes: una

izquierda y otra derecha. Para analizar todas las particiones es necesario examinar todas las subdivisiones de la secuencia de izquierda a derecha por lo tanto hay tantas particiones como puntos en la secuencia menos uno. Si la dirección es diferente de $\frac{\pi}{2}$, se aplica al mismo procedimiento luego de rotar el sistema de coordenadas de forma que la línea que pase por el origen con ángulo α se convierta en el eje "y". Posteriormente se ordenan los puntos en orden no decreciente según el eje de coordenadas x. La nueva coordenada x de los puntos luego de rotar el sistema de coordenadas es $x_v \sin\alpha + y_v \cos\alpha$. A continuación se resume como las particiones son generadas:

Paso 1: Seleccionar un número $N \geq 2$ de direcciones a considerar

Paso 2: Para $i = 1,...,N-1$ calcular $\alpha_i = \frac{\pi}{i}$

Paso 3: Considerar la dirección α_i: ordenar los puntos en V' de forma no decreciente según el valor de $x_v \sin\alpha_i + y_v \cos\alpha_i$. Esta secuencia será denotada por $v_1, v_2, ..., v_s$.

Paso 4: Para $k = 1,2,...,s-1$ examinar la partición dada por $V_i' = \{v_1,...,v_k\}$ y $V_d' = \{v_{k+1},...,v_s\}$.

Los pasos tres y cuatro se repiten para las N direcciones. Se sabe que para $N = 8$ o $N = 16$ se obtienen buenos resultados.

3.1.3.2 ANÁLISIS DE LAS PARTICIONES

El análisis depende de dos factores: balance y compacidad.

3.1.3.2.1 BALANCE

Dado que no se pueden obtener territorios con la misma medida de actividad, se trata de acercarse lo más posible a un balance perfecto. En la secuencia de $v_1, v_2, ..., v_s$ del paso 3 se determina un índice k tal que

$$W(\{v_1, ..., v_{k-1}\}) < \left(\frac{p'_i}{p'}\right) W(V') \quad \text{y}$$

$$W(\{v_1, ..., v_k\}) \geq \left(\frac{p'_i}{p'}\right) W(V')$$

Y se considerará sólo el valor de k que cumpla con la condición anterior que determinará su factibilidad. Además una partición será infactible si $k < p'_i$ o $s - k < p'_d$ o cuando $\frac{W(V'_i)}{p'_i}$ o $\frac{W(V'_d)}{p'_d}$ no es un intervalo $[L, U]$.

3.1.3.2.2 COMPACIDAD

Cada partición es generada por una línea L. El segmento de línea que cae dentro de V' será parte de los límites del territorio en el diseño final, si se trata de que este segmento sea corto se llegará a tener un diseño compacto. Se

utilizarán dos métodos para definir la longitud del segmento.

El primero consiste en emplear la longitud de la intersección de L con la celda convexa C de V', por convexidad si v_k está dentro de C se tiene que L intercepta a C en dos puntos. La distancia Euclidiana entre esos dos puntos define la longitud del segmento.

Si v_k es un vértice de C la longitud puede ser cero.

El uso de celdas convexas es conveniente cuando los puntos o unidades básicas están distribuidos de forma uniforme; pero si no lo están no es conveniente usar este método.

Si los puntos no están distribuidos uniformemente se consideran los puntos de V' que están cercanos a L de tal forma que pertenezcan a una franja definida en base a un porcentaje del ancho de la celda C. Estos puntos se proyectan sobre L y en base a la mayor distancia entre ellos se define la longitud del segmento.

En ambos casos es necesario conocer los puntos que conforman parte de la envoltura convexa, para lo cual se ha

seleccionado el algoritmo de Graham, que se describe a continuación.

3.1.3.3 ALGORITMO DE GRAHAM

La búsqueda de Graham nos permite obtener la celda convexa, dado un conjunto finito de puntos en el plano, con un tiempo de ejecución $O(n\log n)$. Encuentra todos los vértices que conforman la celda convexa. En primer lugar se encuentra el punto o unidad básica con la menor coordenada y, en caso de existir más de un punto con la menor coordenada y. Se busca el punto con la menor coordenada x, al cual se denominará P. Este proceso tiene una complejidad $O(n)$, siendo n el número de puntos a examinar. A continuación los puntos son ordenados según el ángulo formado por los mismos con el punto P y el eje x. Para lo cual no es necesario calcular el ángulo en mención, es suficiente calcular la cotangente de dicho ángulo, la misma que es una función que decrece monótonamente en el dominio $[0,\pi]$, y es calculada en base a las coordenadas de los puntos.

Los puntos se ordenan en forma descendente según el valor de la cotangente, lo que equivale a ordenarlos ascendentemente según su ángulo con el eje x. Una vez ordenados, y siguiendo la secuencia ascendente, se determina si moviéndose desde los dos puntos anteriores al punto actual se ha dado un giro hacia la izquierda o hacia la derecha en el plano cartesiano. Si el giro es hacia

la derecha, se tiene que el penúltimo punto no es parte de la celda convexa y no debe ser considerado. Caso contrario, de presentarse un caso en el que giro es hacia la izquierda, el penúltimo punto examinado es parte de la celda convexa. El proceso continúa hasta que los últimos 3 puntos son examinados. De presentarse el caso de tener puntos colineales, los 3 serán considerados como parte de la celda convexa.

Con la finalidad de determinar de forma analítica si se tiene un giro a la derecha o a la izquierda, no se requiere calcular el ángulo entre los dos segmentos formados por los 3 puntos examinados en cada iteración. Es suficiente para los tres puntos (x_1, y_1), (x_2, y_2) y (x_3, y_3), calcular la dirección del producto cruz de los dos vectores definidos por (x_1, y_1), (x_2, y_2) y (x_1, y_1), (x_3, y_3). Lo cual se obtiene por el signo de:

$$(x_2 - x_1)(y_3 - y_1) - (y_2 - y_1)(x_3 - x_1)$$

Si el resultado es 0, los puntos son colineales, si es positivo se ha efectuado un giro a la izquierda, caso contrario un giro a la derecha. El algoritmo termina en el punto en el que se empezó, es decir el punto P seleccionado inicialmente. La salida del algoritmo está constituida por los puntos pertenecientes a la celda convexa **[10].**

3.1.3.4 ÁRBOL BINARIO DE BÚSQUEDA

El procedimiento voraz para encontrar la mejor partición no es suficiente. A pesar de que en un nivel del árbol se encuentre la mejor partición balanceada, nada garantiza que en el siguiente nivel ocurrirá lo mismo o que $\frac{W(V)'}{p'}$ esté dentro del intervalo $[L,U]$. Para lo cual, el algoritmo incluye un mecanismo de búsqueda hacia atrás, lo que permite volver a visitar un problema básico en el nivel anterior o superior y revisar la partición ejecutada allí, con la finalidad de evitar la infactibilidad.

Se define a u nodo como $\varphi = (V_\varphi, p_\varphi)$, que representa un problema básico a resolver, donde V_φ es el conjunto de unidades básicas a particionar en p_φ territorios. Además cada nodo puede tener tres estados, activo, inactivo y EsHoja. En primer lugar con el problema original se define el nodo raíz del árbol binario a recorrer, y será denotado por $\varphi = (V, p)$, y a su vez, se define su estado como activo.

En cada iteración un nodo activo $\varphi = (V_\varphi, p_\varphi)$ es seleccionado, en caso de que $p_\varphi = 1$, el estado del nodo, se asigna como EsHoja. Caso contrario es un nodo inactivo, y mediante la partición factible mejor ranqueada según el criterio de compacidad, y que no haya sido seleccionada en una iteración anterior, se generan dos nuevos nodos activos $\varphi_l = (V_{\varphi l}, p_{\varphi l})$ y $\varphi_r = (V_{\varphi r}, p_{\varphi r})$. En los cuales, $V_{\varphi l}$ representa el

conjunto de unidades básicas situadas a la izquierda de la línea divisoria, y $V_{\varphi r}$ las unidades básicas de la derecha. El siguiente paso consiste en calcular todas las particiones factibles de los dos problemas generados φ_l y φ_r, y ranquearlas según el criterio de compacidad.

La búsqueda en el árbol binario termina cuando no existen nodos activos y el conjunto de nodos hoja, corresponde al diseño territorial. Se debe tomar en cuenta la posibilidad de que para un nodo activo φ, su problema básico asociado no tenga particiones factibles, por lo que una operación de búsqueda o examinación hacia atrás se debe llevar a cabo. En la cual el nodo padre de φ, cuyo estado es inactivo, se convierte a activo y se eliminan todos sus descendientes, continuando la búsqueda desd este nodo. En caso que la búsqueda hacia atrás, nos conduzca al nodo raíz del árbol, habremos detectado que el problema no es factible con los parámetros, por lo que una estrategia consistiría en relajar las cotas L y U.

3.1.3.5 LIMITACIÓN DEL TAMAÑO DEL ÁRBOL DE BÚSQUEDA

Dado el tipo de árbol, se tiene que durante la búsqueda se examinarán por lo menos $2P-1$ nodos activos, hasta finalizar el proceso de diseñar P territorios. Pero debido a la operación de búsqueda

hacia atrás en caso de infactibilidad, tal cota se puede elevar exponencialmente. Debido a tal situación es necesario limitar la búsqueda, una posibilidad sería detener las iteraciones luego de examinar un número máximo de nodos, y luego mostrar si se llegó o no a una solución factible con los parámetros previamente establecidos. En caso de llegar a un resultado de infactibilidad, se disminuye el valor de L y se incrementa U, de forma que se incremente el número de particiones factibles. No es necesario reiniciar la búsqueda en el árbol binario, sino continuarla con los nuevos parámetros desde el nodo actual. Tal forma de relajación, en caso de ser necesario se repetirá un número definido de ocasiones.

Algoritmo de Heurística de Dicotomía Sucesiva

Entrada: Conjunto de unidades básicas V con sus correspondientes medidas de actividad w_v, $v \in V$, y número de territorios p. Parámetro τ.

Paso 1 Inicialización

Calcular los valores $L = (1-\tau)\frac{W(V)}{p}$ y $U = (1+\tau)\frac{W(V)}{p}$. Se establece el estado del nodo raíz $\varphi_0 = (V, p)$ a activo. Calcular y ranquear todas las particiones factibles según su medida de compacidad.

Paso2: Mientras existan nodos activos en el árbol se hace lo siguiente

Dado un nodo activo $\varphi = (V_\varphi, p_\varphi)$.

Si $p_\varphi = 1$

Establecer estado de nodo φ a EsHoja. Continuar al Paso 2.

Si no hay más particiones factibles para φ

Si $\varphi = \varphi_0$ estamos en el nodo raíz

Relajar L y U, calcular y ranquear todas las particiones factibles de φ_0. Continuar al Paso 2.

Sino

Establecer a activo el estado del nodo padre φ_f de φ y eliminar todos los descendientes de φ_f. Continuar al Paso 2.

Implementar la partición mejor ranqueada mediante la creación de dos nuevos nodos activos $\varphi_l = (V_{\varphi l}, p_{\varphi l})$ y $\varphi_r = (V_{\varphi r}, p_{\varphi r})$. Eliminar esta partición de la lista de particiones de φ. Calcular y ranquear todas las particiones factibles de φ_l y φ_r.

Establecer el estado de φ a inactivo.

Si se excede el número de nodos explorados

Relajar L y U.

Salida

Diseño de territorios definido por los nodos EsHoja

El procedimiento heurístico descrito en esta sección, está basado en **[9]**.

CAPÍTULO 4

4. EL PROBLEMA RUTEO DE VEHÍCULOS

El problema de distribuir productos desde ciertos depósitos a sus usuarios finales juega un papel central en la gestión de algunos sistemas logísticos, y su adecuada planificación puede significar considerables ahorros. Esos potenciales ahorros justifican en gran medida la utilización de técnicas de investigación operativa como facilitadoras de la planificación, dado que se estima que los costos del transporte representan entre el 10% y el 20% del costo final de los bienes **[11].**

Un problema de ruteo de vehículos comprende, dado un conjunto de clientes y depósitos con sus correspondientes coordenadas geográficas, y una flota de vehículos, determinar las rutas que empiecen y culminen en los depósitos, y que a un costo mínimo visiten a cada uno de los clientes. Existen varios escenarios determinados por las características de clientes, depósitos y vehículos, que a su vez definen diversos tipos de problemas de ruteo de vehículos.

Cabe resaltar que la forma de resolver el problema tratado en el presente estudio, establece no abordarlo directamente como un modelo de ruteo de vehículos tomando como parámetros la flota de vehículos, clientes y el depósito; sino en primer lugar elaborar un diseño territorial y luego, dentro de cada territorio resolver el ruteo de los vehículos de reparto. Esto se debe a razones inherentes a estrategias comerciales de atención al mercado. Adicionalmente el alcance en lo relacionado al ruteo comprende la determinación de la secuencia óptima de visita, sin tomar en cuenta características adicionales de clientes y vehículo, dicho problema es conocido como TSP, por sus siglas en inglés (Travelling Salesman Problem), o problema del agente viajero. Dichas características definen

diversas variantes del problema de ruteo, pero no son parte del presente trabajo de investigación.

El interés en el estudio de los problemas de ruteo, no sólo está relacionado a la amplia gama de aplicaciones que poseen en situaciones reales, sino a temas relacionados a la complejidad de su solución. Estos problemas pertenecen a la clase NP-Hard, es decir la motivación académica en resolverlos radica en que no es posible construir algoritmos que en tiempo polinomial resuelvan cualquier instancia del problema, excepto en el caso que P = NP **[9]**.

4.1. MODELO MATEMÁTICO PARA EL TSP

La red de transporte, es decir las calles de la ciudad, o perímetro urbano donde circulan los vehículos, se modela mediante un grafo ponderado $G = (V, E, C)$. Los nodos corresponden a los clientes y depósitos, los cuales conforman el conjunto V. Cada arco representa $(i, j) \in E$ representa el mejor camino desde el nodo i al nodo j con un costo asociado $c_{ij} \in C$. El grafo puede ser simétrico o asimétrico dependiendo de la estructura de costos. Adicionalmente se asume que G es un grafo completo, esto es consistente con una red de transporte real, en la cual es posible ir desde un lugar, o nodo hacia otro. Los conjuntos de nodos adyacentes e incidentes del nodo i se denotarán de la siguiente manera:

$$\Delta^{+}(i) = \{j \in V / (i, j) \in E\} \text{ y } \Delta^{-}(i) = \{j \in V / (j, i) \in E\}$$

En base a lo anterior se tiene el problema del agente viajero con un solo vehículo, el cual debe visitar a todos los clientes que tenga asignada en una jornada. A continuación se presenta una formulación propuesta por Dantzig, Fulkerson y Jonson **[12].**

Minimizar

$$\sum_{(i,j)\in E} c_{ij} x_{ij}$$

Sujeto a:

$$\sum_{j\in\Delta+(i)} x_{ij} = 1 \qquad \forall i \in V$$

$$\sum_{i\in\Delta-(j)} x_{ij} = 1 \qquad \forall j \in V$$

$$\sum_{i\in S,\, j\in\Delta+(i)-S} x_{ij} \geq 1 \qquad \forall S \in V$$

$$x_{ij} \in \{0,1\} \qquad \forall (i,j) \in E$$

Las variables binarias x_{ij} indican si el arco (i,j) pertenece o no a la solución. Por lo que la función objetivo expresa el costo total asociado a los arcos que forman parte de la secuencia óptima de visita, en caso de que el problema sea factible. Los dos primeros conjuntos de restricciones establecen que cada nodo, debe ser visitado y abandonado una sola vez. El último conjunto de restricciones, denominado de eliminación de sub-tours, afirma que todo subconjunto de nodos S debe ser abandonado por lo menos una vez. Esto elimina la posibilidad de existencia de ciclos en la solución.

4.2. HEURÍSTICAS PARA EL TSP

Las heurísticas son métodos que no garantizan la consecución o alcance de una solución óptima, pero si la obtención de buenas soluciones en un tiempo computacional razonable. Existen dos tipos de heurísticas, las que encuentran una solución factible, simplemente partiendo del grafo inicial, y las que mejoran una solución existente.

A continuación se describirán dos heurísticas que parten de cero hasta hallar una solución.

4.2.1 HEURÍSTICA DEL VECINO MÁS PRÓXIMO

El procedimiento comienza en cualquier nodo del grafo, se visita el nodo más cercano que no ha sido visitado, finalmente se retorna al nodo inicial cuando todos los demás nodos ya han sido visitados. Jonson, McGeoch y Rothberg **[13]** demostraron en un problema que la heurística producía una solución 1.26 veces la solución óptima. Lo cual, a pesar de no garantizar la efectividad del procedimiento, indica que es posible obtener buenos resultados en la práctica.

Con la finalidad de garantizar un límite en la desviación respecto a la solución óptima, hay que asumir que los costos asociados a los arcos son no negativos y satisfacen la desigualdad del triángulo **[14],** como sigue a continuación:

$$c_{ij} + c_{jk} \leq c_{ik} \quad \forall i, j, k \in V$$

En caso de cumplirse el caso anterior, Rosenkrants, Stearns y Lewis **[15]** demostraron que el algoritmo del vecino más cercano no sobrepasa en $\frac{1}{2}\lceil \log_2 n \rceil + \frac{1}{2}$ veces la solución óptima. Pero tal cota, no permite cuantificar en la mayoría de los casos la efectividad real de aplicar la heurística a un problema de optimización.

4.2.2 HEURISTICAS DE INSERCIÓN

Las heurísticas de inserción son métodos constructivos en los cuales se crea una solución mediante sucesivas inserciones

de clientes en las rutas. En cada iteración se tiene una solución parcial cuyas rutas sólo visitan un subconjunto de los clientes y se selecciona un cliente no visitado para insertar en dicha solución. Existen diversas variantes, según el tipo de criterio a utilizar previo la inserción de un nodo. A continuación se describen algunas:

4.2.2.1 INSERCIÓN MÁS LEJANA

El algoritmo comienza mediante la selección de un tour que pasa por dos arcos, y que a su vez el costo del arco asociado sea elevado. Luego en cada iteración, para cada nodo v, no perteneciente al tour original, se calcula el mínimo costo entre v y algún nodo del tour construido hasta ese instante. Posteriormente se selecciona el nodo con el máximo costo, y se inserta al tour. El tour resultado del algoritmo a más de ser obtenido en poco tiempo, requiere pocas modificaciones posteriores.

4.2.2.2 INSERCIÓN MÁS CERCANA

Esta heurística es igual a la anterior hasta el paso de seleccionar los nodos candidatos a ser parte del tour de la solución. Se diferencia en el hecho de elegir al nodo que incremente en menor medida el costo del tour.

CAPÍTULO 5

5. IMPLEMENTACIÓN DE MODELO MATEMÁTICO Y ALGORITMOS

5.1. INTRODUCCIÓN A SCIP

Según su acceso y licenciamiento existen dos tipos de solvers para resolver los problemas de programación lineal. Los de uso comercial y no comercial. Para resolver el modelo matemático de diseño de territorios presentado en la sección 2.1.2, emplearemos SCIP (Solving Constraint Integer Programming), uno de los solvers de uso académico más rápidos existentes. Como su nombre lo indica está basado en programación entera de restricciones, para una mayor profundización de dicha teoría, referirse a la Tesis Doctoral de Tobias Achterberg **[16]**.

Tanto para problemas de programación entera y programación de restricciones se usa una técnica similar, dividir el problema original en varios subproblemas, es decir ramificar y luego resolverlos recursivamente. Pero se diferencian en el hecho de que la programación entera usa relajaciones de las restricciones y planos cortantes, y la programación de restricciones puede emplear restricciones no lineales o añadir restricciones al dominio de las variables.

SCIP está implementado en C, por lo que sus métodos pueden ser llamados desde programas de C o C++, y puede resolver problemas dados en formatos MPS y LP **[17]**.

A continuación se presenta un cuadro comparativo del rendimiento de SCIP, tomado de página web de SCIP **[18].**

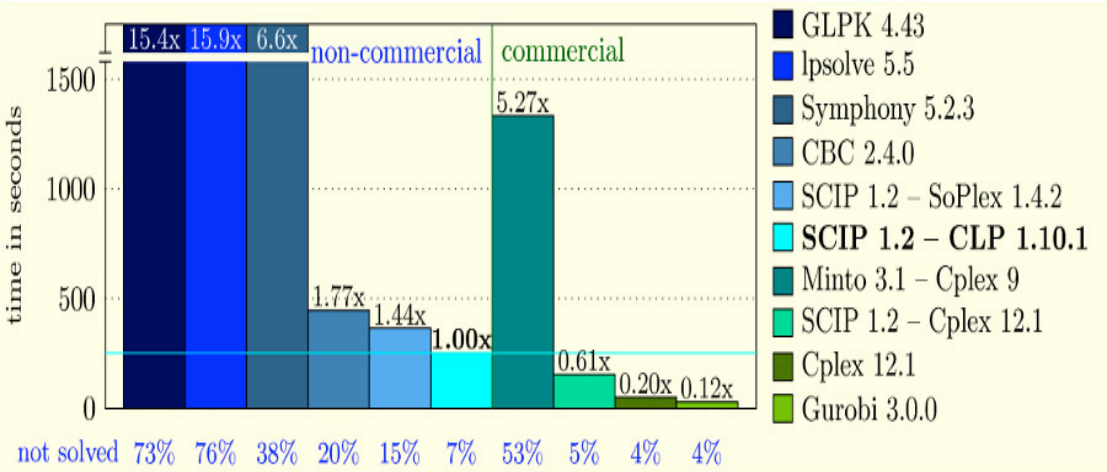


Fig.5.1.1 Cuadro comparativo de Redimiendo de Solvers

5.1.1 IMPLEMENTACIÓN DE MODELO DE DISEÑO DE TERRITORIOS EN SCIP.

SCIP necesita un solver lineal para resolver los programas enteros, en este caso se utilizará SOPLEX, desarrollado por Wunderling Roland, como parte de su tesis doctoral **[19].** Cabe resaltar que el algoritmo fue implementado en Visual Studio C++ .net, para lo cual fue necesario integrar tanto SCIP y SOPLEX en una solución de Visual Studio .net, para lo cual se recurrió a un trabajo de Cornelius Schwarz **[20],** quien en su sitio web provee una solución en Visual Studio, con las interfases necesarias para integrar los dos solvers mencionados. Cabe resaltar que ante las dudas en la implementación Cornelius Schwarz proporcionó el soporte necesario para superar cualquier obstáculo en la integración de los solvers a la plataforma Microsoft.

En la siguiente figura se muestran dos tablas, una denominada tbUnidadesBasicas, en la que se almacena la información referente a cada unidad básica, definida en el

problema de diseño de territorios y la segunda, que es llenada en base a la primera, corresponde a las variables del modelo.

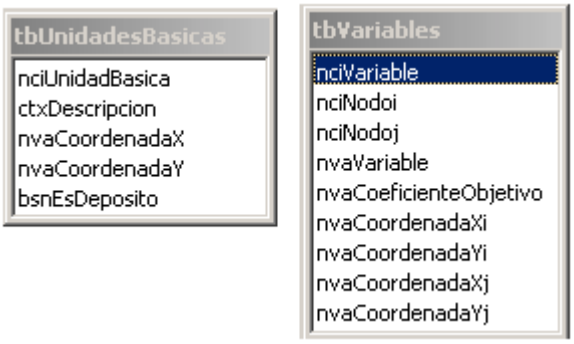


Fig. 5.1.1.1 Tablas de Unidades básicas y Variables

El algoritmo implementado en Visual Studio C++ .net, consta de tres funciones, cuyo objetivo es el siguiente:

- Crear las variables a partir de la tabla de unidades básicas.
- Escritura del archivo LP, con los datos del modelo.
- Programa Principal que ejecuta los métodos de SCIP para obtener una solución.

La implementación de los tres métodos se presenta a continuación:

Creación de Variables

```
  private: System::Void
btCrearArchivo_Click(System::Object *  sender,
System::EventArgs *  e)
  {
  OleDbConnection *cn;
  OleDbCommand *cmd;
  OleDbDataReader *rec;
  cn = new OleDbConnection("Jet OLEDB:Global
Partial Bulk Ops=2;Jet OLEDB:Registry
```

```
Path=;Jet OLEDB:Database Locking Mode=1;Data
Source=C:\\Tesis_Programa\\Database.mdb;Mode=S
hare Deny None;Jet OLEDB:Engine
Type=5;Provider=Microsoft.Jet.OLEDB.4.0;Jet
OLEDB:System database=;Jet
OLEDB:SFP=False;persist security
info=False;Extended Properties=;Jet
OLEDB:Compact Without Replica Repair=False;Jet
OLEDB:Encrypt Database=False;Jet OLEDB:Create
System Database=False;Jet OLEDB:Don't Copy
Locale on Compact=False;User ID=Admin;Jet
OLEDB:Global Bulk Transactions=1");
   cmd = new OleDbCommand();
   cmd->Connection = cn;
   cn->Open();
   int
IDUBDesde,IDUBHasta,IDCriterioDesde,IDCriterio
Hasta,i,j,k;
   IDUBDesde =0;
   IDUBHasta = 0;
   IDCriterioDesde = 0;
   IDCriterioHasta = 0;
     try
   {
   cmd->CommandText = "select
min(nciUnidadBasica),max(nciUnidadBasica) from
tbUnidadesBasicas " ;
   rec = cmd-
>ExecuteReader(Data::CommandBehavior::SingleRo
w);
   if(rec->Read())
   {
        IDUBDesde = rec->GetInt32(0);
```

```
IDUBHasta = rec->GetInt32(1);
  }
  rec->Close();

  k = 0;
  //Creación de Variables en Base de datos
  cmd->CommandText = "DELETE FROM
tbVariables";
  cmd->ExecuteNonQuery();
  cmd->CommandText = "INSERT INTO
tbVariables(nciVariable,nciNodoi,nciNodoj,nvaV
ariable,nvaCoeficienteObjetivo)
values(?,?,?,0,0)";
  cmd->Parameters->Add("@nciVariable"
,OleDbType::Integer);
  cmd->Parameters->Add("@nciNodoi"
,OleDbType::Integer);
  cmd->Parameters->Add("@nciNodoj"
,OleDbType::Integer);

  for(i=IDUBDesde;i<=IDUBHasta;i++)
  {
       for(j=IDUBDesde;j<=IDUBHasta;j++)
       {
            cmd->Parameters-
>get_Item("@nciVariable")->Value = __box(k);
            cmd->Parameters-
>get_Item("@nciNodoi")->Value = __box(i);
            cmd->Parameters-
>get_Item("@nciNodoj")->Value = __box(j);
            cmd->ExecuteNonQuery();
            k = k + 1;
       }
```

```
    }
    //Cálculo de coeficientes de variables en
función objetivo.
    cmd->Parameters->Clear();
    cmd->CommandText = "UPDATE tbVariables INNER
JOIN tbUnidadesBasicas on tbVariables.nciNodoi
= tbUnidadesBasicas.nciUnidadBasica SET
tbVariables.nvaCoordenadaXi =
tbUnidadesBasicas.nvaCoordenadaX,tbVariables.n
vaCoordenadaYi =
tbUnidadesBasicas.nvaCoordenadaY";
    cmd->ExecuteNonQuery();

    cmd->CommandText = "UPDATE tbVariables INNER
JOIN tbUnidadesBasicas on tbVariables.nciNodoj
= tbUnidadesBasicas.nciUnidadBasica SET
tbVariables.nvaCoordenadaXj =
tbUnidadesBasicas.nvaCoordenadaX,tbVariables.n
vaCoordenadaYj =
tbUnidadesBasicas.nvaCoordenadaY";
    cmd->ExecuteNonQuery();

    cmd->CommandText = "UPDATE tbVariables SET
nvaCoeficienteObjetivo = SQR((nvaCoordenadaXi
- nvaCoordenadaXj)*(nvaCoordenadaXi -
nvaCoordenadaXj)+ (nvaCoordenadaYi -
nvaCoordenadaYj)*(nvaCoordenadaYi -
nvaCoordenadaYj))";
    cmd->ExecuteNonQuery();
      }
      catch(Exception *ex)
      {
          MessageBox::Show(   ex->Message);
```

```
 }
  /*cn->Close();
  cn->Dispose();
  cmd->Dispose();  */
  CrearArchivoLP();
  }
```

Escritura del Archivo LP

```
private: void CrearArchivoLP()
{
StreamWriter *escritor;
escritor = new StreamWriter(S"C:\\Solver
SCIP\\Cornelius\\VisualCPP\\SCIP\\scip\\Diseno
Territorios.lp",false);
OleDbConnection *cn;
int
IDUBDesde,IDUBHasta,i,j,a,IDVar,NumeroTerritor
ios,CRIDeste,CRIHasta,Criterio,IDVarDesde,IDVa
rHasta;//Contador de restricciones
double
ValorCriterio,Tolerancia,PromedioCriterio;
Tolerancia = 0;
ValorCriterio = 0;
IDUBDesde = 0;
IDUBHasta = 0;
i = 0;
j = 0;
a = 0;
IDVar = 0;
CRIDeste = 0;
CRIHasta = 0;
Criterio = 0;
```

```
IDVarDesde = 0;
IDVarHasta = 0;
NumeroTerritorios = 0;
NumeroTerritorios = 4;
Tolerancia = 0.30;
PromedioCriterio = 0;

OleDbCommand *cmd;
OleDbDataReader *rec;
cn = new OleDbConnection("Jet OLEDB:Global
Partial Bulk Ops=2;Jet OLEDB:Registry
Path=;Jet OLEDB:Database Locking Mode=1;Data
Source=C:\\Tesis_Programa\\Database.mdb;Mode=S
hare Deny None;Jet OLEDB:Engine
Type=5;Provider=Microsoft.Jet.OLEDB.4.0;Jet
OLEDB:System database=;Jet
OLEDB:SFP=False;persist security
info=False;Extended Properties=;Jet
OLEDB:Compact Without Replica Repair=False;Jet
OLEDB:Encrypt Database=False;Jet OLEDB:Create
System Database=False;Jet OLEDB:Don't Copy
Locale on Compact=False;User ID=Admin;Jet
OLEDB:Global Bulk Transactions=1");
cmd = new OleDbCommand();
cmd->Connection = cn;
cn->Open();


escritor->Write("Minimize ");
cmd->CommandText = "select
nciVariable,nvaCoeficienteObjetivo FROM
tbVariables " ;
```

```
rec = cmd-
>ExecuteReader(Data::CommandBehavior::SingleRe
sult);

while(rec->Read())
{
  escritor->Write(String::Concat(" +", rec-
>GetDouble(1).ToString() ,"x_", rec-
>GetInt32(0).ToString()));
}
rec->Close();
escritor->WriteLine("");
escritor->WriteLine("");
escritor->WriteLine("Subject to");
escritor->WriteLine("");

cmd->CommandText = "select
min(nciUnidadBasica),max(nciUnidadBasica) from
tbUnidadesBasicas " ;
rec = cmd-
>ExecuteReader(Data::CommandBehavior::SingleRo
w);
if(rec->Read())
{
  IDUBDesde = rec->GetInt32(0);
  IDUBHasta = rec->GetInt32(1);
}
rec->Close();


//RESTRICCION:
//CADA UNIDAD BÁSICA PERTENECE A UN SOLO
TERRITORIO
```

```
cmd->CommandText = "select nciVariable FROM
tbVariables WHERE nciNodoi = ? and nciNodoj =
?" ;
cmd->Parameters-
>Add("@nciNodoi",OleDbType::Integer);
cmd->Parameters-
>Add("@nciNodoj",OleDbType::Integer);

a = 0;
for(j=IDUBDesde;j<= IDUBHasta;j++)
{
  escritor->Write(String::Concat("row_",
a.ToString() ,": "));
  for(i=IDUBDesde;i<= IDUBHasta;i++)
  {
       cmd->Parameters->get_Item("@nciNodoi")-
>Value = __box(i);
       cmd->Parameters->get_Item("@nciNodoj")-
>Value = __box(j);
       rec = cmd-
>ExecuteReader(Data::CommandBehavior::SingleRo
w);
       IDVar = -1;
       if(rec->Read())
       {
            IDVar = rec->GetInt32(0);
       }
       rec->Close();
       escritor->Write(String::Concat("+1x_",
IDVar.ToString() ," "));
  }
       escritor->Write("= +1");
       escritor->WriteLine("");
```

```
        a = a + 1;
}

//RESTRICCION:
//CREACION DE P TERRITORIOS
cmd->Parameters->Clear();
cmd->CommandText = "select nciVariable FROM
tbVariables WHERE nciNodoi = nciNodoj and
nciNodoj = ?" ;
cmd->Parameters-
>Add("@nciNodoi",OleDbType::Integer);

escritor->Write(String::Concat("row_",
a.ToString() ,": "));

for(i=IDUBDesde;i<= IDUBHasta;i++)
{
        cmd->Parameters->get_Item("@nciNodoi")-
>Value = __box(i);
        rec = cmd-
>ExecuteReader(Data::CommandBehavior::SingleRo
w);
        IDVar = -1;
        if(rec->Read())
        {
             IDVar = rec->GetInt32(0);
        }
        rec->Close();
        escritor->Write(String::Concat("+1x_",
IDVar.ToString() ," "));
}
  escritor->Write(String::Concat(" =
",NumeroTerritorios.ToString()));
  escritor->WriteLine("");
```

```
  a = a + 1;

//RESTRICCION:
//LIMITE INFERIOR DE TOLERANCIA DE TERRITORIOS
cmd->Parameters->Clear();
cmd->CommandText = "select
min(nciCriterio),max(nciCriterio) from
tbCriterios where bsnActivo = 1" ;
rec = cmd-
>ExecuteReader(Data::CommandBehavior::SingleRo
w);
if(rec->Read())
{
  CRIDeste = rec->GetInt16(0);
  CRIHasta = rec->GetInt16(1);
}
rec->Close();


for(Criterio = CRIDeste; Criterio <=
CRIHasta;Criterio++)
{
  cmd->Parameters->Clear();
  cmd->CommandText = "select nvaPromedio FROM
tbCriterios WHERE nciCriterio = ?" ;
  cmd->Parameters-
>Add("@nciCriterio",OleDbType::Integer);
  cmd->Parameters->get_Item("@nciCriterio")-
>Value = __box(Criterio);
  rec = cmd-
>ExecuteReader(Data::CommandBehavior::SingleRo
w);
```

```
   PromedioCriterio = 0;
   if(rec->Read())
   {
        PromedioCriterio = rec->GetDouble(0);
   }
   rec->Close();

   for(i=IDUBDesde;i<= IDUBHasta;i++)
   {
        escritor->Write(String::Concat("row_",
a.ToString() ,": "));
   for(j=IDUBDesde;j<= IDUBHasta;j++)
   {
        cmd->Parameters->Clear();
        cmd->CommandText = "select nciVariable
FROM tbVariables WHERE nciNodoi = ? and
nciNodoj = ?" ;
        cmd->Parameters-
>Add("@nciNodoi",OleDbType::Integer);
        cmd->Parameters-
>Add("@nciNodoj",OleDbType::Integer);
        cmd->Parameters->get_Item("@nciNodoi")-
>Value = __box(i);
        cmd->Parameters->get_Item("@nciNodoj")-
>Value = __box(j);
        rec = cmd-
>ExecuteReader(Data::CommandBehavior::SingleRo
w);
        IDVar = -1;
        if(rec->Read())
        {
                IDVar = rec->GetInt32(0);
```

```
        }
        rec->Close();

        cmd->Parameters->Clear();
        cmd->CommandText = "select nvaCriterio
FROM tbCriteriosXUnidadBasica WHERE
nciUnidadBasica = ? and nciCriterio = ?" ;
        cmd->Parameters-
>Add("@nciUnidadBasica",OleDbType::Integer);
        cmd->Parameters-
>Add("@nciCriterio",OleDbType::Integer);
        cmd->Parameters-
>get_Item("@nciUnidadBasica")->Value =
__box(j);
        cmd->Parameters-
>get_Item("@nciCriterio")->Value =
__box(Criterio);

        rec = cmd-
>ExecuteReader(Data::CommandBehavior::SingleRo
w);
        ValorCriterio = 0;
        if(rec->Read())
        {
                ValorCriterio = rec-
>GetDouble(0);
        }
        rec->Close();
        escritor->Write(String::Concat("
+",ValorCriterio.ToString() ,"x_",
IDVar.ToString() ));
        }
        cmd->Parameters->Clear();
```

```
cmd->CommandText = "select nciVariable FROM
tbVariables WHERE nciNodoi = nciNodoj and
nciNodoi = ?" ;
        cmd->Parameters-
>Add("@nciNodoi",OleDbType::Integer);
        cmd->Parameters->get_Item("@nciNodoi")-
>Value = __box(i);
        rec = cmd-
>ExecuteReader(Data::CommandBehavior::SingleRo
w);
        IDVar = -1;
        if(rec->Read())
        {
             IDVar = rec->GetInt32(0);
        }
        rec->Close();

        escritor->Write(String::Concat(" -
",PromedioCriterio.ToString() ,"x_",
IDVar.ToString() ));
        escritor->Write(String::Concat(" -
",(PromedioCriterio * Tolerancia).ToString()
,"x_", IDVar.ToString() ));
        escritor->Write(" <= 0");
        escritor->WriteLine("");
        a = a + 1;
        }
}

//RESTRICCION:
//LIMITE SUPERIOR DE TOLERANCIA DE TERRITORIOS
```

```
for(Criterio = CRIDeste; Criterio <=
CRIHasta;Criterio++)
{
cmd->Parameters->Clear();
cmd->CommandText = "select nvaPromedio FROM
tbCriterios WHERE nciCriterio = ?" ;
cmd->Parameters-
>Add("@nciCriterio",OleDbType::Integer);
cmd->Parameters->get_Item("@nciCriterio")-
>Value = __box(Criterio);
rec = cmd-
>ExecuteReader(Data::CommandBehavior::SingleRo
w);
PromedioCriterio = 0;
if(rec->Read())
{
  PromedioCriterio = rec->GetDouble(0);
}
rec->Close();

for(i=IDUBDesde;i<= IDUBHasta;i++)
{
  escritor->Write(String::Concat("row_",
a.ToString() ,": "));
  for(j=IDUBDesde;j<= IDUBHasta;j++)
  {
            cmd->Parameters->Clear();
            cmd->CommandText = "select
nciVariable FROM tbVariables WHERE nciNodoi =
? and nciNodoj = ?" ;
            cmd->Parameters-
>Add("@nciNodoi",OleDbType::Integer);
```

```
                cmd->Parameters-
>Add("@nciNodoj",OleDbType::Integer);


                cmd->Parameters-
>get_Item("@nciNodoi")->Value = __box(i);
                cmd->Parameters-
>get_Item("@nciNodoj")->Value = __box(j);
                rec = cmd-
>ExecuteReader(Data::CommandBehavior::SingleRo
w);
                IDVar = -1;
                if(rec->Read())
                {
                     IDVar = rec->GetInt32(0);
                }
                rec->Close();

                cmd->Parameters->Clear();
                cmd->CommandText = "select
nvaCriterio FROM tbCriteriosXUnidadBasica
WHERE nciUnidadBasica = ? and nciCriterio = ?"
;
                cmd->Parameters-
>Add("@nciUnidadBasica",OleDbType::Integer);
                cmd->Parameters-
>Add("@nciCriterio",OleDbType::Integer);
                cmd->Parameters-
>get_Item("@nciUnidadBasica")->Value =
__box(j);
                cmd->Parameters-
>get_Item("@nciCriterio")->Value =
__box(Criterio);
```

```
        rec = cmd-
>ExecuteReader(Data::CommandBehavior::SingleRo
w);
            ValorCriterio = 0;
            if(rec->Read())
            {
                    ValorCriterio = rec-
>GetDouble(0);
            }
            rec->Close();
            escritor->Write(String::Concat("
+",ValorCriterio.ToString() ,"x_",
IDVar.ToString() ));
        }

        cmd->Parameters->Clear();
        cmd->CommandText = "select nciVariable
FROM tbVariables WHERE nciNodoi = nciNodoj and
nciNodoi = ?" ;
        cmd->Parameters-
>Add("@nciNodoi",OleDbType::Integer);
        cmd->Parameters->get_Item("@nciNodoi")-
>Value = __box(i);
        rec = cmd-
>ExecuteReader(Data::CommandBehavior::SingleRo
w);
        IDVar = -1;
        if(rec->Read())
        {
            IDVar = rec->GetInt32(0);
        }
        rec->Close();
```

```
            escritor->Write(String::Concat(" -
",PromedioCriterio.ToString() ,"x_",
IDVar.ToString() ));
            escritor->Write(String::Concat("
+",(PromedioCriterio * Tolerancia).ToString()
,"x_", IDVar.ToString() ));
            escritor->Write(" >= 0");
            escritor->WriteLine("");
            a = a + 1;
      }

}

escritor->WriteLine("");
escritor->WriteLine("Bounds");

cmd->Parameters->Clear();
cmd->CommandText = "select
min(nciVariable),max(nciVariable) from
tbVariables " ;
rec = cmd-
>ExecuteReader(Data::CommandBehavior::SingleRo
w);
if(rec->Read())
{
  IDVarDesde = rec->GetInt32(0);
  IDVarHasta = rec->GetInt32(1);
}
rec->Close();


for(i = IDVarDesde;i<=IDVarHasta;i++)
{
```

```
  escritor->WriteLine(String::Concat(" +0 <=
x_",i.ToString() ," <= +1 "));
}
escritor->WriteLine("");
escritor->WriteLine("Integer");
escritor->Write(" ");
for(i = IDVarDesde;i<=IDVarHasta;i++)
{
  escritor->Write(String::Concat("
x_",i.ToString()));
}
escritor->WriteLine("");
escritor->WriteLine("End");


 escritor->Close();
 escritor = NULL;
 cn->Close();
 cn->Dispose();
cmd->Dispose();
MessageBox::Show("Archivo Creado...!");
}
```

SCIP programa principal

```
#include <stdio.h>
#include<vector>
#include<sstream>
#include<iostream>
#include<exception>
#include <io.h>
```

```
#include <stdlib.h>
#include<objscip/objscip.h>
#include<objscip/objscipdefplugins.h>

using namespace std;

extern "C"
{
#include "scip/scip.h"
#include "scip/scipshell.h"
}



SCIP_RETCODE runTESIS()
{
   // Crear e inicializar el ambiente de SCIP

FILE *resultado;

//SCIP_File *resultado;

int i=0;
SCIP* scip;
SCIP_CALL(SCIPcreate(& scip));

// Cargar los plugins estándares: corre SCIP
como funcionaría desde una terminal
SCIP_CALL(SCIPincludeDefaultPlugins(scip));
// Cargar parámetros de confirguración desde
el archivo "scipmip.set"
SCIP_CALL(SCIPreadParams(scip,
"scipmip.set"));
```

```
// Crear un nuevo IP
// El primer parámetro es el puntero al
ambiente SCIP
// El segundo es el nombre del problema
("DTerritorios")
// Los demás parámetros sirven para
especificar funciones tipo call back
// y pueden ponerse a NULL sin riesgo
SCIP_CALL(SCIPcreateProb(scip, "SSet1", NULL,
NULL, NULL, NULL, NULL, NULL));

// Especificar el sentido de optimización:
minimizar
SCIP_CALL(SCIPsetObjsense(scip,
SCIP_OBJSENSE_MINIMIZE  ));

//A partir del archivo en formado lp
DisenoTerritorios.lp se crea el problema en
memoria
SCIPreadProb(scip,"DisenoTerritorios.lp");

//Se invoca al método que permite resolver el
problema
SCIP_CALL(SCIPsolve(scip));

//Del conjunto de soluciones se proporciona la
mejor ranqueada.
SCIP_Sol *sol = SCIPgetBestSol(scip);

SCIP_CALL(SCIPwriteMIP(scip,
"DisenoTerritorios1.lp", TRUE, TRUE));

SCIP_VAR** vars;
```

```
//En un arreglo de variables se almacena la
mejor solución encontrada.
vars = SCIPgetOrigVars(scip);

//Si la solución es Nula, terminó el
procedimiento
if(sol==NULL)
{
  // Terminar con mensaje de de error:
  //printf("No fue posible resolver el IP");
  return SCIP_OKAY;
}
else
{

printf( "Solucion: \n" );

double valor = 0;
string cadena;

resultado = fopen( "Resultado.txt", "w" );
valor = SCIPgetSolOrigObj(scip,sol);

// Mostrar valores de cada variable
// IMPORTANTE: Debe tenerse cuidado con las
variables enteras o binarias,
// pues SCIP las maneja a otas internamente
como tipo double
// y puede ser necesario ajustar "manualmente"
los valores
// finales
// Los valores de las variables son almacenado
en el archivo Resultado.txt
```

```
for( int i=0; i<256; i++)
{
  valor = SCIPgetSolVal(scip, sol, vars[i]);

  fprintf(resultado,"%d %e \n",i,valor);

}
}

// Liberar memoria
// Es necesario liberar todas las variables y
restricciones creadas,
// llamando a SCIPreleaseVar y SCIPreleaseCons
// Finalmente, puede liberarse el ambiente
SCIP:
SCIP_CALL(SCIPfree(& scip));
// Se cierra el archivo de texto
fclose(resultado);
return SCIP_OKAY;
}

//Programa Principal
int main(int  argc,char**  argv)
{

SCIP_RETCODE retcode;
retcode = runTESIS();
return retcode;
}
```

5.2. IMPLEMENTACIÓN DE HEURÍSTICA GEOMÉTRICA EN VISUAL STUDIO .NET C#.

La heurística geométrica se implementó en Visual C# .net y para el procesamiento de los datos se empleó una base de datos en Access con las siguientes tablas:

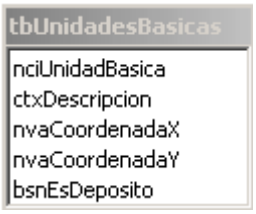


Figura 5.2.1 Tabla de Unidades Básicas

tbUnidadesBasicas.- Contiene las unidades básicas, con sus correspondientes coordenadas.

Figura 5.2.2 Tabla de Nodos Arbol-Tabla de Estados Nodo

tbNodosArbol.- Almacena el árbol binario de búsqueda, cada registro representa un nodo del árbol, con su correspondiente información.

tbEstadosNodo contiene los tres estados posibles de un nodo: Activo, Inactivo y EsHoja(parte de la solución).

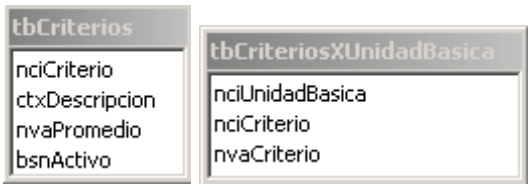


Figura 5.2.3 Tabla de Criterios – Tabla de Criterios por Unidad Básica

tbCriterios contiene los criterios a considerar para el balance de los territorios, por ejemplo beneficios económicos y compacidad. Por otro lado, tbCriteriosXUnidadBasica almacena el valor de los criterios calculados para cada unidad básica.

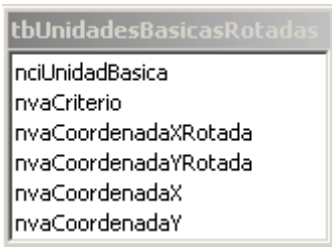


Figura 5.2.4 Tabla de Unidades Básicas Rotadas

tbUnidadesBasicasRotadas.- Esta tabla permite almacenar temporalmente las coordenadas de las unidades básicas rotadas, durante la comprobación del balance entre particiones.

Figura 5.2.5 Tabla de Particiones Factibles – Tabla de Ranking Particiones

Se compone de cinco funciones cuya descripción y objetivos se presentan a continuación:

- HeuristicaGeometrica.- Es la función principal y su misión es construir y examinar el árbol binario de búsqueda. Termina su ejecución cuando se ha encontrado un diseño territorial o bajo los parámetros establecidos no existe una solución factible.
- CalcularYRanquearParticionesFactibles.- Si al visitar un nodo del árbol, este no ha sido particionado, éste método se encarga de determinar las particiones factibles y calcular la medida de compacidad de cada una, con la finalidad de ranquearlas.
- ParticionEsFactible.- Es invocada por la función anterior, y dada una partición devuelve un estado que indica si la partición es o no factible, y cuáles unidades básicas pertenecen a cada partición.
- EliminarDescendencia.- Se invoca en caso de que un nodo no posea una solución factible, y se encarga de eliminar la descendencia del nodo padre, del actualmente examinado.
- ProcesarSolucion.- Almacena en las tablas tbCabeceraSolucionHeur y tbDetallesSolucionHeur los nodos hoja del árbol, que corresponden al diseño territorial.

A continuación se presenta la implementación de la heurística con sus respectivos comentarios.

```
public static void HeuristicaGeometrica(double
FactorRelajacion,double TotalCriterio,double
LimiteInferiorCriterio,double LimiteSuperiorCriterio,double
ToleranciaCriterio,short NumeroTerritorios,short
NumeroDirecciones,string RutaBase)
          {

               int
NumeroTerritoriosNodo,NumeroTerritoriosIzquierdo,NumeroTerr
itoriosDerecho;
               bool ExistenNodosActivos;
```

```
                long
Nodo,NodoPadre,NumPartFactGeneradas,NumPartFactDisponibles,
NodoPadreParticion ;
                long
MejorParticionRankeada,IDNodoIzquierdo,IDNodoDerecho;
                long IDParticionNodo;
                string LadoNodo;
                object obj;
                OleDbTransaction trans;
                OleDbCommand cmd;
                OleDbConnection cn;
                OleDbDataReader rec;

                rec = null;
                cn = new OleDbConnection();
                cmd = new OleDbCommand();



                NodoPadre = 0;
                NodoPadreParticion = 0;


                try
                {
                     //Se limpian las Tablas.
                     cn.ConnectionString = RutaBase;
                     cn.Open();
                     cmd.Connection = cn;
                     cmd.CommandText = "delete from
tbNodosArbol" ;
                     cmd.ExecuteNonQuery();


                     cmd.CommandText = "delete from
tbParticionesFactibles";
                     cmd.ExecuteNonQuery();

                     cmd.CommandText = "delete from
tbRankingParticiones";
                     cmd.ExecuteNonQuery();

                     cmd.CommandText = "insert into
tbNodosArbol(nciNodo,nciNodoPadre,ncaTerritorios,nciEstado,
cciLado,nnuParticion) values(1,0," +
NumeroTerritorios.ToString() + ",1,'',1)";
                     cmd.ExecuteNonQuery();

                     ExistenNodosActivos = true;
```

```
                    NumeroTerritoriosNodo =
NumeroTerritorios;

                    Nodo= 1;
                    //Mientras existan nodos activos se
examinará el árbol binario de búsqueda
                    while(ExistenNodosActivos )
                    {
                         //Si el nodo se debe dividir en un
solo territorio, se asigna su estado a Eshoja
                         //y se continúa a la verificación
del while
                         cmd.CommandText = "select
nciNodo,ncaTerritorios,cciLado,nnuParticion,nciNodoPadre
from tbNodosArbol where nciEstado = 1";
                         rec =
cmd.ExecuteReader(CommandBehavior.SingleRow);
                         Nodo = 0;

                         Nodo = -1;
                         NumeroTerritoriosNodo = 0;
                         LadoNodo = "";
                         IDParticionNodo = 0;
                         if(rec.Read())
                         {
                                   Nodo = rec.GetInt32(0);
                                   NumeroTerritoriosNodo=
rec.GetInt16(1);
                                   LadoNodo =
rec.GetString(2).Trim() ;
                                   IDParticionNodo =
rec.GetInt32(3);
                                   NodoPadreParticion =
rec.GetInt32(4);
                         }
                         rec.Close();



                         //En caso de existir un nodo
activo un nodo activo
                         if( Nodo > 0)
                         {
                              ExistenNodosActivos = true;

                              NumPartFactGeneradas=0;
                              //Si el número de territorios
en que el nodo se debe particionar es mayor a uno se
continúa
```

```
                                    //caso contrario se le asigna
su estado a EsNodoHoja y es parte de la solución
                                    if(NumeroTerritoriosNodo > 1)
                                    {

                                         //Determinar si Nodo ha
generado particiones
                                         cmd.CommandText =
"select count(1) from tbRankingParticiones where
nciNodoPadre = " + Nodo.ToString();
                                         NumPartFactGeneradas =
(Int32 ) cmd.ExecuteScalar();

                                         NumPartFactDisponibles =
0;
                                         if(NumPartFactGeneradas
== 0)
                                         {
                                              //Con los
paràmetros actuales no se han generado particiones de Nodo
                                              //Se procede a
generar particiones por primera vez

     NumPartFactDisponibles =
CalcularYRanquearParticionesFactibles(RutaBase,NumeroDirecc
iones
,NumeroTerritoriosNodo,LimiteInferiorCriterio,LimiteSuperio
rCriterio,Nodo,LadoNodo,IDParticionNodo,NodoPadreParticion)
;
                                         }
                                         else
                                         {
                                              //Ya se han
generado particiones factibles anteriormente
                                              //no es la primera
vez qeu se visita el nodo
                                              cmd.CommandText =
"select count(1) from tbRankingParticiones where bsnElegido
= 0 and nciNodoPadre = " + Nodo.ToString();

     NumPartFactDisponibles = (Int32) cmd.ExecuteScalar();
                                         }


     if(NumPartFactDisponibles ==0)
                                         {
                                              // no hay
particiones factibles para el nodo: Nodo Padre
```

```
                                        if(Nodo==1)// por
lo que el nodo infactible es el nodo raíz
                                        {
                                             //se procede a
relajar las cotas

     ToleranciaCriterio = ToleranciaCriterio +
FactorRelajacion;

     LimiteInferiorCriterio = (1-
ToleranciaCriterio)*TotalCriterio/NumeroTerritorios;

     LimiteSuperiorCriterio =
(1+ToleranciaCriterio)*TotalCriterio/NumeroTerritorios;
                                             //se deben
limpiar las tablas relacionadas al arbol de búsqueda de
soluciones


     cmd.Transaction = cn.BeginTransaction();

     cmd.CommandText = "delete from  tbNodosArbol where
nciNodo <> 1";

     cmd.ExecuteNonQuery();


     cmd.CommandText = "update tbNodosArbol set nciEstado =
1,nnuMejorParticionFactible = 0";

     cmd.ExecuteNonQuery();


     cmd.CommandText = "delete from
tbParticionesFactibles";

     cmd.ExecuteNonQuery();


     cmd.CommandText = "delete from tbRankingParticiones";

     cmd.ExecuteNonQuery();

     cmd.Transaction.Commit();



     ExistenNodosActivos = true;
```

```
                                        }
                                        else
                                        {
                                            //se actualiza
el esdado del nodo padre de Nodo, a activo
                                            NodoPadre = 0;

     cmd.CommandText = "select nciNodoPadre from
tbNodosArbol where nciNodo = " + Nodo.ToString()  ;
                                            NodoPadre
=(Int32 ) cmd.ExecuteScalar();


     cmd.CommandText = "update tbNodosArbol set nciEstado =
1,nnuMejorParticionFactible = 0 where nciNodo = " +
NodoPadre.ToString() ;

     cmd.ExecuteNonQuery();
                                            //se eliminan
todos los descendientes de nodo padre


     EliminarDescendencia(NodoPadre,cmd,cn);


                                        }

                                    }
                                    else
                                    {

                                        //se actualiza el
estado del nodo padre a inactivo
                                        cmd.CommandText =
"update tbNodosArbol set nciEstado = 2 where nciNodo = " +
Nodo.ToString();

     cmd.ExecuteNonQuery();

                                        //se implementa la
mejor partición no seleccionada.

     MejorParticionRankeada = 0;


                                        cmd.CommandText =
"select top 1 nnuParticionFactible from
tbRankingParticiones where bsnElegido = 0 and nciNodoPadre
= " + Nodo.ToString() +
```

```
          " and nvaMedidaCompacidad in (select
     min(nvaMedidaCompacidad) from tbRankingParticiones
     where bsnElegido = 0 and nciNodoPadre = " +
     Nodo.ToString() + ")";



     MejorParticionRankeada = (Int32 ) cmd.ExecuteScalar();
                                       IDNodoIzquierdo =
0;
                                       IDNodoDerecho = 0;

     NumeroTerritoriosIzquierdo = 0;

     NumeroTerritoriosDerecho = 0;

     if(MejorParticionRankeada!=0)
                                       {
                                            //Se actualiza
el estado de la partición a elegido

     cmd.Transaction = cn.BeginTransaction();

     cmd.CommandText = "update tbRankingParticiones set
bsnElegido = 1 where nciNodoPadre = " + Nodo.ToString() + "
and nnuParticionFactible = " +
MejorParticionRankeada.ToString() ;

     cmd.ExecuteNonQuery();

     cmd.Transaction.Commit();

                                       //Se añaden
los nodos al árbol
                                       //ID de nuevos
nodos

     cmd.CommandText = "select max(nciNodo) from
tbNodosArbol";

     IDNodoIzquierdo = (Int32) cmd.ExecuteScalar() ;

     IDNodoIzquierdo = IDNodoIzquierdo + 1;
                                       IDNodoDerecho
= IDNodoIzquierdo + 1;

                                       //Número de
Territorios de nuevos nodos
                                       //Número
territorios izquierdo
```

```
     cmd.CommandText = "select top 1 ncaTerritorios from
tbParticionesFactibles " +
                                                  "where
cciLado = 'I' and nciNodoPadre = " + Nodo.ToString() + "
and nnuParticionFactible = " +
MejorParticionRankeada.ToString()  ;
                                                  obj =
cmd.ExecuteScalar();

                                                  if(obj!=null)
                                                  {

     NumeroTerritoriosIzquierdo = (Int16) obj;

                                                  }



                                                  //Número
territorios izquierdo

     cmd.CommandText = "select top 1 ncaTerritorios from
tbParticionesFactibles " +
                                                  "where
cciLado = 'D' and nciNodoPadre = " + Nodo.ToString() + "
and nnuParticionFactible = " +
MejorParticionRankeada.ToString()  ;
                                                  obj =
cmd.ExecuteScalar();

                                                  if(obj!=null)
                                                  {

     NumeroTerritoriosDerecho = (Int16) obj;

                                                  }



                                                  //Se insertan
los nodos hijos de Nodo
                                                  //Nodo
izquierdo
                                                  trans =
cn.BeginTransaction();

     cmd.Transaction = trans;

     cmd.CommandText = "insert into
```

```
tbNodosArbol(nciNodo,nciNodoPadre,ncaTerritorios,nciEstado,
nnuMejorParticionFactible,cciLado,nnuParticion) " +
                                                "values(" +
IDNodoIzquierdo.ToString()    + "," + Nodo.ToString() + ","
+ NumeroTerritoriosIzquierdo.ToString() + ",1,0,'I'," +
MejorParticionRankeada.ToString() + ")";

     cmd.ExecuteNonQuery();

                                                //Nodo Derecho

     cmd.CommandText = "insert into
tbNodosArbol(nciNodo,nciNodoPadre,ncaTerritorios,nciEstado,
nnuMejorParticionFactible,cciLado,nnuParticion) " +
                                                "values(" +
IDNodoDerecho.ToString()    + "," + Nodo.ToString() + "," +
NumeroTerritoriosDerecho.ToString() + ",1,0,'D'," +
MejorParticionRankeada.ToString() + ")";

     cmd.ExecuteNonQuery();

     cmd.Transaction.Commit();

                                                //se debe
actualizar en el nodo actual del árbol

     cmd.CommandText = "update tbNodosArbol set
nnuMejorParticionFactible = " +
MejorParticionRankeada.ToString() + " where nciNodo = " +
Nodo.ToString();

     cmd.ExecuteNonQuery();



                                          }


                                       }

//////////////////////////////////////////


                                    }
                                    else
                                    {

     if(NumeroTerritoriosNodo==1)
```

```
                                        {
                                              cmd.CommandText =
"update tbNodosArbol set nciEstado = 3 where nciNodo = " +
Nodo.ToString();

     cmd.ExecuteNonQuery();
                                        }

                                   }
                              }
                              else
                              {
                                   ExistenNodosActivos = false;

                              }

                    }

                         //actualizar los nodos del árbol y
continuar con el proceso mientras existan nodos activos.


                    }
                    catch(Exception ex)
                    {
                    MessageBox.Show(ex.Message,"Heurística
Geométrica");
                    }
                    finally
                    {


                    }
               }



public static int CalcularYRanquearParticionesFactibles(string
RutaBase,short NumeroDirecciones,int NumeroTerritorios,double
LimiteInferiorCriterio,double LimiteSuperiorCriterio, long
NodoPadre,string LadoNodo ,long IDParticionFactible,long
NodoPadreParticion)
               {
                    int
NumeroTerritoriosIzquierdo,NumeroTerritoriosDerecho;

                    OleDbCommand cmd;
                    OleDbConnection cn;
                    OleDbDataReader rec;
```

```
                    rec = null;
                    cn = new OleDbConnection();
                    cmd = new OleDbCommand();

                    cn.ConnectionString = RutaBase;
                    cn.Open();
                    cmd.Connection = cn;
                    short i;
                    ArrayList _ListaParticiones;
              int
j,NumeroParticionFactible,m,EsNodokGraham,NumeroParticionesFactibles;
                    long NodoK;
                    double
TotalCriterioConjunto,ValorCompacidad,CoordenadaXNodoK,CoordenadaYNodo
K;
                    double
resulDBL,PXCoordenadaGraham,PYCoordenadaGraham;
                    double
DireccionGiro,AnguloParticionFactible,PendienteSuperior,PendienteInfer
ior,Angulo;
                    double
InterseccionXSuperior,InterseccionYSuperior,InterseccionXInferior,Inte
rseccionYInferior;
                    object obj;

                    string IDLadoPGraham = string.Empty;
                    InterseccionXSuperior= 0;
                    InterseccionXInferior = 0;

                    _ListaParticiones = null;

                    long IDParticion,IDUnidadBasica ;
                    CoordenadaXNodoK = 0;
                    bool EsParticionFactible;
                    object valor;
                    Nodo _Nodo,_Nodo1,_Nodo2,_Nodo3;
                    Nodo _NodoIzMin,_NodoIzMax,_NodoDeMin,_NodoDeMax;
                    Coleccion _ListaNodos,_ListaNodosGraham;
                    _ListaNodos = new Coleccion();
                    _ListaNodosGraham  = new Coleccion();

                    _NodoIzMin = null;
                    _NodoIzMax = null;
                    _NodoDeMin = null;
                    _NodoDeMax = null;
                    //comentar
                    NumeroDirecciones = 0;
                          for(i=0;i<=NumeroDirecciones;i++)
                          {

                                if(i> 0)
                                      Angulo = System.Math.PI/i;
                                else
                                      Angulo = 0;

                                cmd.CommandText = "delete from
tbUnidadesBasicasRotadas";
                                cmd.ExecuteNonQuery();
```

```
                        // se guardan unidades básicas rotadas al
nuevo eje de coordenadas
                        cmd.Transaction = cn.BeginTransaction();
                        if(NodoPadre > 1)
                        {
                              cmd.CommandText = "insert into
tbUnidadesBasicasRotadas(nciUnidadBasica,nvaCriterio,nvaCoordenadaXRot
ada,nvaCoordenadaYRotada,nvaCoordenadaX,nvaCoordenadaY) " +
                                    "select
nciUnidadBasica,nvaCriterio,nvaCoordenadaX * cos(" + Angulo.ToString()
+ ") - nvaCoordenadaY * sin(" + Angulo.ToString()   +
"),nvaCoordenadaX * sin(" + Angulo.ToString()   + ") + nvaCoordenadaY
* cos(" + Angulo.ToString()   + "),nvaCoordenadaX,nvaCoordenadaY " +
                                    "from tbParticionesFactibles
where nciNodoPadre = " + NodoPadreParticion.ToString() + " and cciLado
= '" + LadoNodo + "' and nnuParticionFactible = " +
IDParticionFactible.ToString() ;
                              cmd.ExecuteNonQuery();
                        }
                        else
                        {
                              cmd.CommandText = "insert into
tbUnidadesBasicasRotadas(nciUnidadBasica,nvaCriterio,nvaCoordenadaXRot
ada,nvaCoordenadaYRotada,nvaCoordenadaX,nvaCoordenadaY) " +
                                    "select
tbUnidadesBasicas.nciUnidadBasica,tbCriteriosXUnidadBasica.nvaCriterio
,tbUnidadesBasicas.nvaCoordenadaX * cos(" + Angulo.ToString()   + ") -
tbUnidadesBasicas.nvaCoordenadaY * sin(" + Angulo.ToString()   + "), "
+

      "tbUnidadesBasicas.nvaCoordenadaX * sin(" + Angulo.ToString()
+ ") + tbUnidadesBasicas.nvaCoordenadaY * cos(" + Angulo.ToString()
+ "),tbUnidadesBasicas.nvaCoordenadaX,tbUnidadesBasicas.nvaCoordenadaY
" +
                                    "from tbUnidadesBasicas inner
join tbCriteriosXUnidadBasica on tbUnidadesBasicas.nciUnidadBasica =
tbCriteriosXUnidadBasica.nciUnidadBasica where
tbCriteriosXUnidadBasica.nciCriterio = 1" ;

                              cmd.ExecuteNonQuery();

                        }
                        cmd.Transaction.Commit();



                        //A continuación se determinará si la
partición es factible
                        cmd.CommandText = "select
nciUnidadBasica,nvaCriterio,nvaCoordenadaXRotada,nvaCoordenadaYRotada
from tbUnidadesBasicasRotadas order by nvaCoordenadaXRotada asc";
                        rec = cmd.ExecuteReader();
                        _ListaNodos.Clear();

                        while(rec.Read())
                        {
                              _Nodo = new Nodo();
                              _Nodo.IDNodo = rec.GetInt32(0);
                              _Nodo.Criterio1 = rec.GetDouble(1);
```

```
                                    _Nodo.CoordenadaX =
rec.GetDouble(2);
                                    _Nodo.CoordenadaY =
rec.GetDouble(3);
                                    _ListaNodos.Add(_Nodo);
                              }
                              rec.Close();

                              TotalCriterioConjunto = 0;
                              cmd.CommandText = "select
sum(nvaCriterio) as total from tbUnidadesBasicasRotadas";
                              valor = cmd.ExecuteScalar();
                              if(valor != System.DBNull.Value)
                              {
                                    TotalCriterioConjunto =
double.Parse( valor.ToString());
                              }

                              j = 0;

                              EsParticionFactible = false;

                              //Determinar si existe una partición
factible para la dirección Angulo
                              NodoK = 0;
                              NumeroTerritoriosIzquierdo = 0;
                              NumeroTerritoriosDerecho = 0;

                              if(NumeroTerritorios % 2 == 0)
                              {
                                    NumeroTerritoriosIzquierdo =
NumeroTerritorios/2;
                                    NumeroTerritoriosDerecho =
NumeroTerritoriosIzquierdo;
                                    EsParticionFactible =
ParticionEsFactible(_ListaNodos,NumeroTerritoriosIzquierdo,
NumeroTerritoriosDerecho,NumeroTerritorios,TotalCriterioConjunto,Limit
eInferiorCriterio,LimiteSuperiorCriterio,ref NodoK,ref
CoordenadaXNodoK);

                              }
                              else
                              {
                                    NumeroTerritoriosIzquierdo =
int.Parse( decimal.Floor(NumeroTerritorios/2).ToString()) ;
                                    NumeroTerritoriosDerecho  =
NumeroTerritorios - NumeroTerritoriosIzquierdo;
                                    EsParticionFactible =
ParticionEsFactible(_ListaNodos,NumeroTerritoriosIzquierdo,
NumeroTerritoriosDerecho,NumeroTerritorios,TotalCriterioConjunto,Limit
eInferiorCriterio,LimiteSuperiorCriterio,ref NodoK,ref
CoordenadaXNodoK);
                                    if(!EsParticionFactible)
                                    {
                                          NumeroTerritoriosIzquierdo =
NumeroTerritoriosIzquierdo + 1;
                                          NumeroTerritoriosDerecho  =
NumeroTerritorios - NumeroTerritoriosIzquierdo;
```

```
EsParticionFactible =
ParticionEsFactible(_ListaNodos,NumeroTerritoriosIzquierdo,
NumeroTerritoriosDerecho,NumeroTerritorios,TotalCriterioConjunto,Limit
eInferiorCriterio,LimiteSuperiorCriterio,ref NodoK,ref
CoordenadaXNodoK);

                                    }
                              }

                              //En caso de existir la dirección
factible para Ángulo
                              if(EsParticionFactible)
                              {
                                    NumeroParticionFactible = 0;
                                    cmd.CommandText = "select
max(nnuParticionFactible) from  tbParticionesFactibles where
nciNodoPadre = " +  NodoPadre.ToString();
                                    valor = cmd.ExecuteScalar();
                                    if(valor!=DBNull.Value)
                                          NumeroParticionFactible =
int.Parse(valor.ToString());
                                    NumeroParticionFactible =
NumeroParticionFactible + 1;

                                    cmd.Transaction =
cn.BeginTransaction();
                                    cmd.CommandText = "insert into
tbParticionesFactibles(nciNodoPadre,nnuParticionFactible,nciUnidadBasi
ca,nvaCriterio,nvaAngulo,nvaCoordenadaXRotada,nvaCoordenadaYRotada,cci
Lado,ncaTerritorios,nvaCoordenadaX,nvaCoordenadaY) "  +
                                          "select " +
NodoPadre.ToString() + "," + NumeroParticionFactible.ToString() +
",nciUnidadBasica,nvaCriterio," + Angulo.ToString() +
",nvaCoordenadaXRotada,nvaCoordenadaYRotada, 'I'," +
NumeroTerritoriosIzquierdo.ToString()  +
",nvaCoordenadaX,nvaCoordenadaY from tbUnidadesBasicasRotadas where
nvaCoordenadaXRotada <= " + CoordenadaXNodoK.ToString()  ;
                                    cmd.ExecuteNonQuery();

                                    cmd.CommandText = "insert into
tbParticionesFactibles(nciNodoPadre,nnuParticionFactible,nciUnidadBasi
ca,nvaCriterio,nvaAngulo,nvaCoordenadaXRotada,nvaCoordenadaYRotada,cci
Lado,ncaTerritorios,nvaCoordenadaX,nvaCoordenadaY) "  +
                                          "select " +
NodoPadre.ToString() + "," + NumeroParticionFactible.ToString() +
",nciUnidadBasica,nvaCriterio," + Angulo.ToString() +
",nvaCoordenadaXRotada,nvaCoordenadaYRotada, 'D', " +
NumeroTerritoriosDerecho.ToString()   +
",nvaCoordenadaX,nvaCoordenadaY from tbUnidadesBasicasRotadas where
nvaCoordenadaXRotada > " + CoordenadaXNodoK.ToString()  ;
                                    cmd.ExecuteNonQuery();

                                    cmd.CommandText = "update
tbParticionesFactibles set bsnNodoK = 1 where nciNodoPadre = " +
NodoPadre.ToString() + " and nnuParticionFactible = " +
NumeroParticionFactible.ToString() + " and nciUnidadBasica = "  +
NodoK.ToString() ;
                                    cmd.ExecuteNonQuery();
                                    cmd.Transaction.Commit();
```

```
                              }//if(EsParticionFactible)


                        }//for(i=1;i<=NumeroDirecciones;i++)

                        //Una vez insertadas todas las particiones
factibles, se debe proceder a ranquearlas, para escoger la mejor
                        //se escogen las particiones factibles de nodo
padre
                        cmd.CommandText = "select nnuParticionFactible
from tbParticionesFactibles where nciNodoPadre = " +
NodoPadre.ToString() +
                              " group by nnuParticionFactible";
                        rec = cmd.ExecuteReader();

                        if(_ListaParticiones==null)
                              _ListaParticiones = new ArrayList();
                        else
                              _ListaParticiones.Clear();
                        // se recorren cada una de las particiones
factibles
                        while(rec.Read())
                        {
                              _ListaParticiones.Add(rec.GetInt32(0));
                        }
                        rec.Close();

                        if(_ListaParticiones.Count> 0)
                        {

                              //Para cada partición factible calculo su
medida de compacidad
                              //Previo al calculo de la medida, se debe
determinar los puntos

                              //                        1.- Determinar
para cada particion la celda convexa mediante el algoritmo de Graham.
                              //                        2.- Estimar la
medida de compacidad mediante las dos formas ya sea si
                              //                              los puntos
estàn uniformemente distribuidos o no.
                              //                      3.- Utilizar la
particion mejor ranqueada, las demas quedan guardadas para ser usadas.

                              j = 0;


                              while(j<_ListaParticiones.Count)
                              {
                                    IDParticion =
(Int32)_ListaParticiones[j];
                                    ValorCompacidad = 0;



                                    cmd.CommandText = "delete from
tbConjuntoUBSinOrdenar";
                                    cmd.ExecuteNonQuery();
```

```
                                    cmd.CommandText = "delete from
tbConjuntoUBOrdenado";
                                    cmd.ExecuteNonQuery();

                                    cmd.Parameters.Clear();
                                    cmd.Parameters.Add("@nciNodoPadre"
,OleDbType.BigInt);

      cmd.Parameters.Add("@nnuParticionFactible",  OleDbType.BigInt);
                                    cmd.Parameters[
"@nciNodoPadre"].Value = NodoPadre;
                                    cmd.Parameters[
"@nnuParticionFactible"].Value =IDParticion;
                                    //verificar la menor coordenada y,
si hay un empate empezamos con la X
                                    cmd.CommandText = "select
MIN(nvaCoordenadaYRotada) from tbParticionesFactibles where
nciNodoPadre = ? and nnuParticionFactible = ?";
                                    PXCoordenadaGraham = 0;
                                    PYCoordenadaGraham = 0;
                                    obj = null;
                                    obj = cmd.ExecuteScalar();
                                    resulDBL = 0;
                                    IDUnidadBasica = 0;
                                    if(obj!=null)
                                    {

                                          resulDBL = (double)obj;
                                    }
                                    rec.Close();


                                    //Capturo ángulo de la partición
factible para ser usado más adelante
                                    cmd.CommandText = "select nvaAngulo
from tbParticionesFactibles where nciNodoPadre = ? and
nnuParticionFactible = ? group by nvaAngulo";
                                    obj  = null;
                                    obj = cmd.ExecuteScalar();
                                    AnguloParticionFactible = 0;
                                    if(obj!=null)
                                    {
                                          if(obj!=System.DBNull.Value)
                                                AnguloParticionFactible
=(double) obj  ;
                                    }
                                    rec.Close();



                                    PXCoordenadaGraham = 0;
                                    PYCoordenadaGraham = 0;
                                    IDLadoPGraham = string.Empty;
                                    EsNodokGraham = 0;


                                          cmd.CommandText = "select top
1
nciUnidadBasica,nvaCoordenadaXRotada,nvaCoordenadaYRotada,cciLado,bsnN
```

```
odoK from tbParticionesFactibles where nciNodoPadre = ? and
nnuParticionFactible = ? and nvaCoordenadaYRotada = " +
resulDBL.ToString();
                                        rec =
cmd.ExecuteReader(System.Data.CommandBehavior.SingleRow);
                                    if(rec.Read())
                                    {
                                        IDUnidadBasica =
rec.GetInt32(0);
                                        PXCoordenadaGraham =
rec.GetDouble(1);
                                        PYCoordenadaGraham =
rec.GetDouble(2);
                                        IDLadoPGraham =
rec.GetString(3);
                                        EsNodokGraham =
rec.GetByte(4);
                                    }
                                    rec.Close();


                                    //Calcular la cotangente para el
resto de los puntos a considerar
                                    //inserto el punto P en base al
cual se va a determinar si los puntos restantes ordenados
                                    //están o no ordenados respecto a
la cotangente.

                                    cmd.Parameters.Clear();
                                    cmd.Transaction =
cn.BeginTransaction();
                                    cmd.CommandText = "insert into
tbConjuntoUBSinOrdenar(nciUnidadBasica,nvaCoordenadaX,nvaCoordenadaY,b
snPuntoPGraham,cciLado,bsnNodoK) " +
                                        "values(" +
IDUnidadBasica.ToString() + "," +  PXCoordenadaGraham.ToString() + ","
+  PYCoordenadaGraham.ToString() + ",1,'" + IDLadoPGraham + "'," +
EsNodokGraham.ToString() + ")";
                                    cmd.ExecuteNonQuery();

                                    //Insertar unidades básicas con su
contangente asociada previo a ordenación
                                    cmd.CommandText = "insert into
tbConjuntoUBSinOrdenar(nciUnidadBasica,nvaCoordenadaX,nvaCoordenadaY,b
snPuntoPGraham,nvaCotangente,cciLado,bsnNodoK) " +
                                        "select
nciUnidadBasica,nvaCoordenadaXRotada,nvaCoordenadaYRotada,0,nvaCoorden
adaXRotada/nvaCoordenadaYRotada,cciLado,bsnNodoK " +
                                        "from tbParticionesFactibles
where nciUnidadBasica <> " + IDUnidadBasica.ToString() + " and
nciNodoPadre = " + NodoPadre.ToString()   + " and nnuParticionFactible
= " + IDParticion.ToString()   ;
                                    cmd.ExecuteNonQuery();
```

```
                                    //Insertar en tabla de unidades
básicas rotadas punto P, a partir del cual se ejecutará el algoritmo
de Graham
                                    cmd.CommandText = "insert into
tbConjuntoUBOrdenado(nciUnidadBasica,nvaCoordenadaX,nvaCoordenadaY,bsn
CeldaConvexa,bsnPuntoPGraham,cciLado,bsnNodoK) " +
                                          "values(" +
IDUnidadBasica.ToString() + "," +  PXCoordenadaGraham.ToString() + ","
+  PYCoordenadaGraham.ToString() + ",1,1,'" + IDLadoPGraham + "'," +
EsNodokGraham.ToString() + ")";
                                    cmd.ExecuteNonQuery();

                                    cmd.CommandText = "insert into
tbConjuntoUBOrdenado(nciUnidadBasica,nvaCoordenadaX,nvaCoordenadaY,bsn
CeldaConvexa,bsnPuntoPGraham,nvaCotangente,cciLado,bsnNodoK) " +
                                          "select
nciUnidadBasica,nvaCoordenadaX,nvaCoordenadaY,0,0,nvaCotangente,cciLad
o,bsnNodoK " +
                                          "from tbConjuntoUBSinOrdenar
where nciUnidadBasica <> " + IDUnidadBasica.ToString()  + " order by
nvaCotangente desc";
                                    cmd.ExecuteNonQuery();
                                    cmd.Transaction.Commit();



                                    //En este punto ya se tienen las UB
ordenadas según su cotangente.

                                    //Se guardan las UB en un arreglo
para implementar el algoritmo de Graham
                                    cmd.CommandText = "select
nciUnidadBasica,nvaCoordenadaX,nvaCoordenadaY from
tbConjuntoUBOrdenado";
                                    rec = cmd.ExecuteReader();

                                    _ListaNodosGraham.Clear();

                                    while(rec.Read())
                                    {
                                          _Nodo = new Nodo();
                                          _Nodo.IDNodo =
rec.GetInt32(0);
                                          _Nodo.CoordenadaX =
rec.GetDouble(1);
                                          _Nodo.CoordenadaY =
rec.GetDouble(2);
                                          _ListaNodosGraham.Add(_Nodo);
                                    }
                                    rec.Close();


                                    cmd.Parameters.Clear();
                                    cmd.CommandText = "update
tbConjuntoUBOrdenado set bsnPuntoPGraham = 1 where nciUnidadBasica =
?";

      cmd.Parameters.Add("@nciUnidadBasica",SqlDbType.BigInt);
```

```
if(_ListaNodosGraham.Count > 3)
                                    {
                                          m = 2;

      while(m<=_ListaNodosGraham.Count -1)
                                          {
                                                _Nodo1 =(Nodo)
_ListaNodosGraham[m-2];
                                                _Nodo2 =(Nodo)
_ListaNodosGraham[m-1];
                                                _Nodo3 =(Nodo)
_ListaNodosGraham[m];
                                                DireccionGiro =
(_Nodo2.CoordenadaX - _Nodo1.CoordenadaX)* (_Nodo3.CoordenadaY  -
_Nodo1.CoordenadaY) -

      (_Nodo2.CoordenadaY  - _Nodo1.CoordenadaY)*(_Nodo3.CoordenadaX -
_Nodo1.CoordenadaX);

                                                //Puntos Colineales o
_Nodo2 perteneciente a celda convexa, caso contrario no pertenece y
conserva valor 0
                                                if(DireccionGiro>=0)
                                                {
                                                      cmd.Parameters[
"@nciUnidadBasica"].Value = _Nodo2.IDNodo ;

      cmd.ExecuteNonQuery();
                                                }
                                                m = m + 1;
                                          }
                                          //Hay que considerar si el
último punto pertenece o no a la celda convexa
                                          _Nodo1 =(Nodo)
_ListaNodosGraham[_ListaNodosGraham.Count -2];
                                          _Nodo2 =(Nodo)
_ListaNodosGraham[_ListaNodosGraham.Count -1];
                                          _Nodo3 =(Nodo)
_ListaNodosGraham[0];
                                          DireccionGiro =
(_Nodo2.CoordenadaX - _Nodo1.CoordenadaX)* (_Nodo3.CoordenadaY  -
_Nodo1.CoordenadaY) -
                                                (_Nodo2.CoordenadaY  -
_Nodo1.CoordenadaY)*(_Nodo3.CoordenadaX - _Nodo1.CoordenadaX);

                                          //Puntos Colineales o _Nodo2
perteneciente a celda convexa, caso contrario no pertenece y conserva
valor 0
                                          if(DireccionGiro>=0)
                                          {
                                                cmd.Parameters[
"@nciUnidadBasica"].Value = _Nodo2.IDNodo ;
                                                cmd.ExecuteNonQuery();

                                          }


                                    }//fin  if(_ListaNodosGraham.Count
> 3)
```

```
                                    else
                                    {
                                          if(_ListaNodosGraham.Count ==
3)
                                          {
                                                //la figura sería un
triángulo, por lo que si los puntos no son colineales
                                                //los 3 son la celda
convexa

                                          }



                                    }

                                    _ListaNodosGraham.Clear();

                                    //Hasta aquí tengo los nodos
pertenecientes a la celda convexa
                                    //ahora sigue calcular la medida de
compacidad


                                    // selecciono los nodos máximo y
mínimo por cada lado: izquierdo o derecho
                                    cmd.Parameters.Clear();
                                    cmd.CommandText = "select
nciUnidadBasica,nvaCoordenadaX,nvaCoordenadaY from
tbConjuntoUBOrdenado where cciLado = 'I' order by nvaCoordenadaY
desc";
                                    rec =
cmd.ExecuteReader(CommandBehavior.SingleRow);

                                    //Nodo izquiero máximo
                                    if(rec.Read())
                                    {
                                          _NodoIzMax = new Nodo();
                                          _NodoIzMax.IDNodo =
rec.GetInt32(0);
                                          _NodoIzMax.CoordenadaX =
rec.GetDouble(1);
                                          _NodoIzMax.CoordenadaY =
rec.GetDouble(2);


                                    }
                                    rec.Close();

                                    //Derecho Mäximo
                                    cmd.CommandText = "select
nciUnidadBasica,nvaCoordenadaX,nvaCoordenadaY from
tbConjuntoUBOrdenado where cciLado = 'D' order by nvaCoordenadaY
desc";
                                    rec =
cmd.ExecuteReader(CommandBehavior.SingleRow);

                                    if(rec.Read())
                                    {

                                          _NodoDeMax = new Nodo();
```

```
                                        _NodoDeMax.IDNodo =
rec.GetInt32(0);
                                        _NodoDeMax.CoordenadaX =
rec.GetDouble(1);
                                        _NodoDeMax.CoordenadaY =
rec.GetDouble(2);

                                    }
                                    rec.Close();


                                    cmd.CommandText = "select
nciUnidadBasica,nvaCoordenadaX,nvaCoordenadaY from
tbConjuntoUBOrdenado where cciLado = 'I' order by nvaCoordenadaY asc";
                                    rec =
cmd.ExecuteReader(CommandBehavior.SingleRow);
                                    //Nodo izquiero mínimo
                                    if(rec.Read())
                                    {
                                        _NodoIzMin = new Nodo();
                                        _NodoIzMin.IDNodo =
rec.GetInt32(0);
                                        _NodoIzMin.CoordenadaX =
rec.GetDouble(1);
                                        _NodoIzMin.CoordenadaY =
rec.GetDouble(2);

                                    }
                                    rec.Close();

                                    //Derecho mínimo
                                    cmd.CommandText = "select
nciUnidadBasica,nvaCoordenadaX,nvaCoordenadaY from
tbConjuntoUBOrdenado where cciLado = 'D' order by nvaCoordenadaY asc";
                                    rec =
cmd.ExecuteReader(CommandBehavior.SingleRow);

                                    if(rec.Read())
                                    {
                                        _NodoDeMin  = new Nodo();
                                        _NodoDeMin.IDNodo =
rec.GetInt32(0);
                                        _NodoDeMin.CoordenadaX =
rec.GetDouble(1);
                                        _NodoDeMin.CoordenadaY =
rec.GetDouble(2);

                                    }
                                    rec.Close();


                                    // Con los puntos más altos de cada
lado, se debe calcular la intersección de las rectas
                                    // formadas por la unión de estos
puntos(altos y bajos respectivamente) con la recta L que forma la
partición


                                    //Intersección superior
```

```
                                    //1.- Calcular pendiente de recta
qeu une los dos puntos Superiores.
                                    PendienteSuperior = 0;
                                    PendienteInferior = 0;
                                    InterseccionXSuperior = 0;
                                    InterseccionYSuperior= 0;
                                    InterseccionXInferior = 0;
                                    InterseccionYInferior = 0;
                                    CoordenadaXNodoK = 0;
                                    CoordenadaYNodoK = 0;

                                    cmd.CommandText =  "select
nvaCoordenadaX,nvaCoordenadaY from tbConjuntoUBOrdenado where bsnNodoK
= 1";
                                    rec =
cmd.ExecuteReader(CommandBehavior.SingleRow);
                                    if(rec.Read())
                                    {
                                          CoordenadaXNodoK =
rec.GetDouble(0) ;
                                          CoordenadaYNodoK =
rec.GetDouble(1) ;

                                    }
                                    rec.Close();




                                    if(_NodoDeMax.CoordenadaX -
_NodoIzMax.CoordenadaX != 0)
                                          PendienteSuperior =
(_NodoDeMax.CoordenadaY -
_NodoIzMax.CoordenadaY)/(_NodoDeMax.CoordenadaX  -
_NodoIzMax.CoordenadaX );

                                    if(_NodoDeMin.CoordenadaX -
_NodoIzMin.CoordenadaX != 0)
                                          PendienteInferior =
(_NodoDeMin.CoordenadaY -
_NodoIzMin.CoordenadaY)/(_NodoDeMin.CoordenadaX  -
_NodoIzMin.CoordenadaX );




                                    InterseccionYSuperior =
PendienteSuperior*(CoordenadaXNodoK - _NodoIzMax.CoordenadaX) +
_NodoIzMax.CoordenadaY ;
                                    InterseccionYInferior =
PendienteInferior*(CoordenadaXNodoK - _NodoIzMin.CoordenadaX) +
_NodoIzMin.CoordenadaY ;
                                    ValorCompacidad =
Math.Sqrt(Math.Pow(CoordenadaXNodoK - CoordenadaXNodoK,2) +
Math.Pow(InterseccionYSuperior - InterseccionYInferior  ,2));



                                    cmd.Transaction =
cn.BeginTransaction();
```

```
                                    cmd.CommandText = "insert into
tbRankingParticiones(nciNodoPadre,nnuParticionFactible,nvaAngulo,nvaMe
didaCompacidad,bsnElegido) " +
                                                "values (" +
NodoPadre.ToString()  + "," + IDParticion.ToString() + "," +
AnguloParticionFactible.ToString() + "," + ValorCompacidad.ToString()
+ ",0)";
                                    cmd.ExecuteNonQuery();
                                    cmd.Transaction.Commit();
                                    j = j +1;
                                }//While recorrer particiones




                        }//if(_ListaParticiones.Count> 0)

                        //retornar el número de particiones factibles
del nodo padre

                        NumeroParticionesFactibles = 0;
                        cmd.Parameters.Clear();
                        cmd.CommandText = "select count(1) from
tbRankingParticiones where bsnElegido = 0 and nciNodoPadre = " +
NodoPadre.ToString() ;
                        obj = null;
                        obj = cmd.ExecuteScalar();
                        if(obj!=null)
                        {
                              NumeroParticionesFactibles  = (int)obj;
                        }
                        rec.Close();

                        cn.Close();
                        cn.Dispose();
                        cmd.Dispose();


                  return NumeroParticionesFactibles;

            }


public static bool ParticionEsFactible(Coleccion ListaNodos,int
NumeroTerritoriosIzquierdo,int NumeroTerritoriosDerecho,int
NumeroTerritorios, double TotalCriterioConjunto, double
LimiteInferiorCriterio,double LimiteSuperiorCriterio,ref long
NodoK,ref double CoordenadaNodoK)
            {
            bool Detener = false;
            bool EsParticionFactible = false;
            double TotalCriterioParticion = 0,FactorPromedioCriterio =
0;
            double PromedioCriterioIzquierdo =
0,PromedioCriterioDerecho = 0;
            int k,l ;
                  l = 0;
                  k = 0;
```

```
                    Nodo _Nodo;
                    NodoK = 0;
                    CoordenadaNodoK = 0;
                    while(l<ListaNodos.Count-1 && Detener==false )
                    {
                          _Nodo = (Nodo) ListaNodos[l];
                          TotalCriterioParticion = _Nodo.Criterio1 +
TotalCriterioParticion;
                          FactorPromedioCriterio =
(double.Parse(NumeroTerritoriosIzquierdo.ToString())
/double.Parse(NumeroTerritorios.ToString()) )*TotalCriterioConjunto;
                          if(TotalCriterioParticion <
FactorPromedioCriterio  && (TotalCriterioParticion + ((Nodo)
ListaNodos[l+1]).Criterio1) >= FactorPromedioCriterio )
                          {
                                EsParticionFactible = true;
                                k = l+1;
                                Detener = true;
                                NodoK = _Nodo.IDNodo;
                                CoordenadaNodoK = _Nodo.CoordenadaX;
                          }
                          if(k<NumeroTerritoriosIzquierdo ||
ListaNodos.Count - k<NumeroTerritoriosDerecho)
                          {
                                EsParticionFactible = false;
                                NodoK = 0;
                          }

                          PromedioCriterioIzquierdo =
TotalCriterioParticion/NumeroTerritoriosIzquierdo;

                          if(PromedioCriterioIzquierdo <
LimiteInferiorCriterio || PromedioCriterioIzquierdo >
LimiteSuperiorCriterio)
                          {
                                EsParticionFactible = false;
                                NodoK = 0;
                          }

                          PromedioCriterioDerecho =
(TotalCriterioConjunto -
TotalCriterioParticion)/NumeroTerritoriosDerecho ;
                          if(  PromedioCriterioDerecho <
LimiteInferiorCriterio || PromedioCriterioDerecho >
LimiteSuperiorCriterio)
                          {
                                EsParticionFactible = false;
                                NodoK= 0;
                          }
                          l = l +1 ;

                    }
                    return EsParticionFactible;
              }

public static bool EliminarDescendencia(long NodoPadre,OleDbCommand
cmd,OleDbConnection cn)
              {
```

```
                    Coleccion _ListaNodos;
                    OleDbDataReader rec;
                    Nodo _Nodo;
                    int j;
                    _ListaNodos = new Coleccion();
                cmd.CommandText = "select nciNodo from tbNodosArbol
where nciNodoPadre = " + NodoPadre.ToString();
                    rec =
cmd.ExecuteReader(CommandBehavior.SingleResult);
                    while(rec.Read())
                    {
                    _Nodo = new Nodo();
                          _Nodo.IDNodo = rec.GetInt32(0);
                    _ListaNodos.Add(_Nodo);

                    }
                    rec.Close();

                    j = 0;
                    while(j<_ListaNodos.Count)
                    {
                          _Nodo = (Nodo) _ListaNodos[j];
                          cmd.CommandText = "delete from tbNodosArbol
where nciNodo = "   + _Nodo.IDNodo ;
                          cmd.ExecuteNonQuery();
                          EliminarDescendencia(_Nodo.IDNodo,cmd,cn);

                          j = j +1;
                    }

                          return true;

            }
```

5.3. IMPLEMENTACIÓN DE HEURÍSTICA DE INSERCIÓN PARA EL TSP EN VISUAL STUDIO .NET C#.

La heurística de inserción se implementó en Visual C# .net y para el procesamiento de los datos se emplearon las tablas en las que se almacenarán las soluciones al problema de diseño de territorios comerciales, correspondientes al modelo matemático y a la heurística geométrica. A continuación se presenta una descripción de cada una de ellas.

tbVariables.- En esta estructura se almacena la solución del modelo matemático implementado en SCIP. Para cada uno de los territorios en el campo nnUSecuenciaVisita se colocará la secuencia y en nvaCostoTSP el costo del tour.

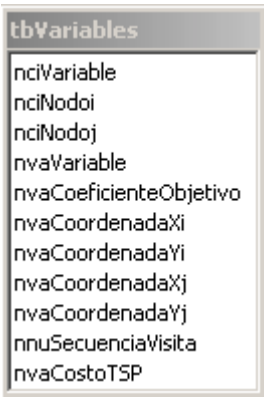


Figura 5.3.1 Tabla de Variables

tbCabeceraSolucionHeur y tbDetallesSolucionHeur.- En estas tablas se guardan los nodos hoja del árbol binario de búsqueda, es decir, las soluciones dadas por la heurística. La tabla cabecera contiene la información de cada territorio: número de unidades básicas, valor del criterio, y costo del tour de visitas. La de detalles, además de la información de cada unidad básica, contiene la secuencia de visita.

Figura 5.3.2 Tabla de Particiones Factibles – Tabla de Ranking Particiones

Según sea el caso, la heurística tomará el conjunto de unidades básicas asignadas a cada territorio en la solución, y determinará una secuencia factible, con un costo cercano al óptimo.

Se compone de tres funciones cuya descripción y objetivos se presentan a continuación:

- EjecutarTSPSolucionScip.- Esta función prepara los datos de la solución del modelo matemático, almacenado en tbVariables y ejecuta el TSP para cada uno de los territorios.
- EjecutarTspSolucionHeuristica.- Prepara los datos de la solución de la heurística geométrica, y ejecuta el TSP para cada uno de los nodos solución del árbol binario de búsqueda.
- ResolverTSPFarthestInsertion.- Recibe como parámetros la lista de unidades básicas pertenecientes a cada territorio, y determina una secuencia razonable de visita.

A continuación se presenta la implementación de los métodos debidamente comentados.

```
private void EjecutarTSPSolucionScip()
            {
                  int i,j,NumeroTerritorios,IDNodoTerritorio;
                  long IDUnidadBasica = 0;
                  ArrayList _ListaTerritorios;
                  IDNodoTerritorio = 0;
                  double XDeposito = 0,YDeposito = 0;
                  double CostoTotal=0;
                  Coleccion _RutaMejorada;
                  Coleccion  _Nodos = new Coleccion();
                  Nodo _Deposito,_nodo;
                  OleDbConnection cn;
                  OleDbCommand cmd;
                  DateTime inicio,fin;
                  TimeSpan duracion;
                  OleDbDataReader rs;
                  _Deposito = null;

                  TSP _tsp;
                  _tsp =  new TSP();
                  try
                  {
                        cn = new OleDbConnection(RutaBase);
                        cn.Open();

                        cmd = new OleDbCommand();
                        cmd.Connection = cn;
```

```
                        //Determinar Número de Territorios a los que se
le va a calcular la secuencia óptima
                        //Cada Territorio se lo ingresa en un ArrayList
y se pasa como parámetro al mètodo de la heurística

                        _ListaTerritorios = new ArrayList();
                        cmd.CommandText = "select nciNodoi from
tbVariables where nvaVariable = 1 group by nciNodoi";
                        rs =
cmd.ExecuteReader(CommandBehavior.SingleResult);
                        while(rs.Read())
                        {
                              _ListaTerritorios.Add(rs.GetInt32(0));
                        }
                        rs.Close();
                        cn.Close();
                        //Para cada territorio en _ListaTerritorios, se
calcula la secuencia óptima
                        i  = 0;

                        //Debe ser un nodo con la mayor coordenada en X
y en Y
                        //Bodega ubicada al nor Este de Guayaquil
                        cn.Open();
                        cmd.CommandText = "select max(nvaCoordenadaXj)
from tbVariables where nvaVariable = 1";
                        XDeposito = (double) cmd.ExecuteScalar();
                        cmd.CommandText = "select max(nvaCoordenadaYj)
from tbVariables where nvaVariable = 1";
                        YDeposito  = (double) cmd.ExecuteScalar();
                        _Deposito = new Nodo();
                        _Deposito.IDNodo = 0;
                        _Deposito.CoordenadaX = XDeposito;
                        _Deposito.CoordenadaY =YDeposito;
                        this.txtCoordenadaXDepositoSCIP.Text=
XDeposito.ToString();
                        this.txtCoordenadaYDepositoSCIP.Text=
YDeposito.ToString();
                        cn.Close();
                        while(i<_ListaTerritorios.Count)
                        {
                              IDNodoTerritorio =(Int32)
_ListaTerritorios[i];
                              _Nodos.Clear();
                              cn.Open();
                              cmd.CommandText = "select
nciNodoj,nvaCoordenadaXj,nvaCoordenadaYj  from tbVariables where
nvaVariable = 1 and nciNodoi = "+ IDNodoTerritorio.ToString();
                              rs =
cmd.ExecuteReader(CommandBehavior.SingleResult);
                              while(rs.Read())
                              {
                                    _nodo = new Nodo();
                                    _nodo.IDNodo = rs.GetInt32(0);
                                    _nodo.CoordenadaX =
rs.GetDouble(1);
                                    _nodo.CoordenadaY =
rs.GetDouble(2);
                                    _Nodos.Add(_nodo);
```

```
                                }
                                rs.Close();
                                cn.Close();
                                inicio = DateTime.Now;
                                _RutaMejorada =
_tsp.ResolverTSPFarthestInsertion(_Nodos,_Deposito);
                                fin = DateTime.Now;
                                duracion = fin - inicio;
                                cn.Open();
                                cmd.CommandText = "update tbVariables set
ncaMinutos = " + duracion.Minutes.ToString()  + ",ncaSegundos = " +
duracion.Seconds  +
                                      " where nvaVariable = 1 and
nciNodoi = " + IDNodoTerritorio.ToString();
                                cmd.ExecuteNonQuery();
                                cn.Close();


                                if(_RutaMejorada!=null)
                                {
                                      j = 0;
                                      CostoTotal = 0;
                                      while(j<_RutaMejorada.Count)
                                      {
                                            IDUnidadBasica =((Nodo)
_RutaMejorada[j]).IDNodo ;
                                            if(j>=1 && j<
_RutaMejorada.Count)
                                            {
                                                  CostoTotal = CostoTotal
+ Nodo.CalcularDistanciaEuclidiana((Nodo) _RutaMejorada[j],(Nodo)
_RutaMejorada[j - 1]);

                                            }
                                            cn.Open();
                                            cmd.CommandText = "update
tbVariables set nnuSecuenciaVisita = " + (j+1).ToString()+ " where
nvaVariable = 1 and nciNodoj = " + IDUnidadBasica.ToString() + " and
nciNodoi = " + IDNodoTerritorio.ToString() ;
                                            cmd.ExecuteNonQuery();
                                            cn.Close();

                                            j = j +1;

                                      }
                                      cn.Open();
                                      cmd.CommandText = "update
tbVariables set nvaCostoTSP = " + CostoTotal.ToString()  + " where
nvaVariable = 1 and nciNodoi = " + IDNodoTerritorio.ToString() ;
                                      cmd.ExecuteNonQuery();
                                      cn.Close();

                                }

                                i = i +1;
                          }
                          cn.Dispose();
                          cmd.Dispose();
```

```
                }
                catch(Exception ex)
                {
                      MessageBox.Show(ex.Message);
                }
                finally
                {

                }
          }



private void EjecutarTSPSolucionHeuristica()
          {
                int i,j,NumeroTerritorios,IDNodoTerritorio;
                long IDUnidadBasica = 0;
                ArrayList _ListaTerritorios;
                IDNodoTerritorio = 0;
                double XDeposito = 0,YDeposito = 0;
                double CostoTotal=0;
                Coleccion _RutaMejorada;
                Coleccion  _Nodos = new Coleccion();
                Nodo _Deposito,_nodo;
                OleDbConnection cn;
                OleDbCommand cmd;
                OleDbDataReader rs;
                _Deposito = null;
                DateTime inicio,fin;
                TimeSpan duracion;

                TSP _tsp;
                _tsp =  new TSP();
                try
                {
                      cn = new OleDbConnection(RutaBase);
                      cn.Open();

                      cmd = new OleDbCommand();
                      cmd.Connection = cn;
                      //Determinar Número de Territorios a los que se
le va a calcular la secuencia óptima
                      //Cada Territorio se lo ingresa en un ArrayList
y se pasa como parámetro al mètodo de la heurística
                      cmd.CommandText = "select count(1) from
tbCabeceraSolucionHeur";
                      NumeroTerritorios = (Int32)
cmd.ExecuteScalar();

                      if(NumeroTerritorios>0)
                      {
                            _ListaTerritorios = new ArrayList();
                            cmd.CommandText = "select nciNodo from
tbCabeceraSolucionHeur";
                            rs =
cmd.ExecuteReader(CommandBehavior.SingleResult);
                            while(rs.Read())
                            {
```

```
      _ListaTerritorios.Add(rs.GetInt32(0));
                              }
                              rs.Close();

                              //Para cada territorio en
_ListaTerritorios, se calcula la secuencia óptima
                              i  = 0;

                              //Debe ser un nodo con la mayor
coordenada en X y en Y
                              //Bodega ubicada al nor Este de Guayaquil
                              cmd.CommandText = "select
max(nvaCoordenadaX) from tbDetallesSolucionHeur";
                              XDeposito = (double) cmd.ExecuteScalar();
                              cmd.CommandText = "select
max(nvaCoordenadaY) from tbDetallesSolucionHeur";
                              YDeposito  = (double)
cmd.ExecuteScalar();
                              cn.Close();
                              _Deposito = new Nodo();
                              _Deposito.IDNodo = 0;
                              _Deposito.CoordenadaX = XDeposito;
                              _Deposito.CoordenadaY =YDeposito;
                              this.txtCoordenadaXDeposito.Text=
XDeposito.ToString();
                              this.txtCoordenadaYDeposito.Text=
YDeposito.ToString();

                              while(i<_ListaTerritorios.Count)
                              {
                                    IDNodoTerritorio =(Int32)
_ListaTerritorios[i];
                                    _Nodos.Clear();
                                    cn.Open();
                                    cmd.CommandText = "select
nciUnidadBasica,nvaCoordenadaX,nvaCoordenadaY  from
tbDetallesSolucionHeur where nciNodo = "+ IDNodoTerritorio.ToString();
                                    rs =
cmd.ExecuteReader(CommandBehavior.SingleResult);
                                    while(rs.Read())
                                    {
                                          _nodo = new Nodo();
                                          _nodo.IDNodo =
rs.GetInt32(0);
                                          _nodo.CoordenadaX =
rs.GetDouble(1);
                                          _nodo.CoordenadaY =
rs.GetDouble(2);
                                          _Nodos.Add(_nodo);

                                    }
                                    rs.Close();
                                    cn.Close();

                                    inicio = DateTime.Now;
                                    _RutaMejorada =
_tsp.ResolverTSPFarthestInsertion(_Nodos,_Deposito);
                                    fin = DateTime.Now;
                                    duracion = fin - inicio;
```

```
                                    cn.Open();
                                    cmd.CommandText = "update
tbCabeceraSolucionHeur set ncaMinutos = " +
duracion.Minutes.ToString()  + ",ncaSegundos = " + duracion.Seconds  +
                                          " where nciNodo = " +
IDNodoTerritorio.ToString();
                                    cmd.ExecuteNonQuery();

                                    cn.Close();

                                    if(_RutaMejorada!=null)
                                    {
                                          j = 0;
                                          CostoTotal = 0;
                                          while(j<_RutaMejorada.Count)
                                          {
                                                IDUnidadBasica =((Nodo)
_RutaMejorada[j]).IDNodo ;
                                                if(j>=1 && j<
_RutaMejorada.Count)
                                                {
                                                      CostoTotal =
CostoTotal + Nodo.CalcularDistanciaEuclidiana((Nodo)
_RutaMejorada[j],(Nodo) _RutaMejorada[j - 1]);

                                                }
                                                cn.Open();
                                                cmd.CommandText =
"update tbDetallesSolucionHeur set nnuSecuenciaVisita = " +
(j+1).ToString()+ " where nciUnidadBasica = " +
IDUnidadBasica.ToString() ;
                                                cmd.ExecuteNonQuery();
                                                cn.Close();

                                                j = j +1;
                                          }
                                          cn.Open();
                                          cmd.CommandText = "update
tbCabeceraSolucionHeur set nvaCostoTSP = " + CostoTotal.ToString()  +
" where nciNodo = " + IDNodoTerritorio.ToString() ;
                                          cmd.ExecuteNonQuery();
                                          cn.Close();

                                    }
                                    i = i +1;
                              }

                        }
                        cn.Close();
                        cn.Dispose();
                        cmd.Dispose();
                        MessageBox.Show("Terminado");

                  }
                  catch(Exception ex)
                  {
                        MessageBox.Show(ex.Message);
                  }
                  finally
                  {
```

```
                }
            }



public Coleccion ResolverTSPFarthestInsertion(Coleccion
ListaNodos,Nodo Deposito)
            {
                  Coleccion Ruta  = new Coleccion();
                  Nodo NodoA,NodoB;
                  Double CostoMaximo,Costo,CostoMinimo,Tope;
                  CostoMaximo = 1000;
                  Costo = 0;
                  //Par de nodos Iniciales
                  // Nodos con al máxima distancia.
                  NodoA = Deposito;
                  Tope = 0;
                  NodoB = null;
                  foreach(Nodo n in ListaNodos)
                  {
                        Costo =
Grafos.Nodo.CalcularDistanciaEuclidiana(Deposito,n);
                        Tope = Tope + Costo;

                        //Costo Mínimo
                        if  (CostoMaximo >= Costo)
                        {
                              CostoMaximo = Costo;
                              NodoB = n;
                        }
                  }
                  // sirve para especificar que NodoB ya pertenece al
tour
                  NodoB.Marcado = true;
                  Ruta.Add(Deposito);
                  Ruta.Add(NodoB);
                  ListaNodos.Remove(NodoB);

                  while(ListaNodos.Count > 0 )
                  {
                        //para cada no perteneciente al tour
                        foreach(Nodo n in ListaNodos)
                        {

                                    CostoMinimo = Tope;
                                    Costo= 0;
                                    //se calcula la distancia mínima a
un nodo  del tour
                                    foreach(Nodo m in Ruta)
                                    {
                                          Costo =
Grafos.Nodo.CalcularDistanciaEuclidiana(m,n);
                                          if  (CostoMinimo > Costo)
                                          {
                                                CostoMinimo = Costo;
                                                NodoB = m;

                                          }
```

```
                                    }
                                    n.NodoRelacionado = new Nodo();

      n.NodoRelacionado.CopiarDatosSimples(NodoB);
                                    NodoB = null;
                              }
                              //escojer entre los nodos
relacionados(perteneciente al tour y no marcados)
                              //la distancia maxima y ese nodo se lo agrega a
la ruta
                              Costo = 0;
                              CostoMaximo = 10000000;
                              foreach(Nodo n in ListaNodos)
                              {
                                    Costo = Math.Sqrt(Math.Pow(n.CoordenadaX
- n.NodoRelacionado.CoordenadaX,2)+Math.Pow(n.CoordenadaY  -
n.NodoRelacionado.CoordenadaY ,2));
                                    //Costo mínimo
                                    if (Costo < CostoMaximo )
                                    {
                                          CostoMaximo = Costo;
                                          NodoB = n;
                                    }
                              }

                              if(NodoB!=null)
                              {
                                    ListaNodos.Remove(NodoB);
                                    Ruta.Add(NodoB);
                              }
                        }
                        return Ruta;
                  }
```

CAPÍTULO 6

6. ANÁLISIS DE RESULTADOS COMPUTACIONALES

En el presente capítulo se presenta varios resultados interesantes sobre el rendimiento de las implementaciones tanto del modelo en Scip, como de la heurística geométrica. Cabe resaltar que los datos empleados son de una empresa de consumo masivo de Guayaquil, y pertenecen a una muestra de 500 puntos de venta. Por efectos de confidencialidad los beneficios económicos de cada punto de venta fueron transformados, con la finalidad de que no reflejen los beneficios reales. Para las prueban se consideraron diferentes números de unidades básicas a zonificar, para observar el desenvolvimiento de los métodos implementados. El número de territorios de la solución se fijó en 16.

En primer lugar analizaremos la solución proporcionada por el **modelo matemático implementado en SCIP.** En la tabla 6.1 observamos, que no se halló una solución óptima, dada la infactibilidad del problema, a medida que se definían tolerancias menores, y según se aumentaban las unidades básicas a zonificar. Además se tiene que el número de variables aumenta de forma exponencial en función del número de unidades básicas. Este tipo de problema pertenece a la clase NP duro.

	Tolerancias						Número
# U. Básicas	0.05	0.10	0.20	0.30	0.40	0.50	Variables
80	NF	NF	1206.035	1198.13	1176.44	1155.46	6400
120	NF	NF	1925.64	1907.16	1896.88	1896.88	14400
200	NF	NF	NF	NF	NF	NF	40000

Figura 6.1 Valor de la solución en SCIP y Número de variables por # de unidades básicas

	Tolerancias			
# U. Básicas	0.20	0.30	0.40	0.50
80	1:21:19	1:24:37	1:47:48	0:49:22
120	4:09:19	4:05:50	4:23:10	3:20:31

Figura 6.2 Tiempo de ejecución de implementación en SCIP (hh:mm:ss)

La tabla 6.2 nos indica que a medida que aumenta el número de unidades básicas, se incrementa el tiempo de ejecución. Lo que ocurre no de forma proporcional, ilustrando el incremento exponencial del tiempo de ejecución. Además se observa que el menor tiempo de ejecución, corresponde a la mayor tolerancia para la medida del criterio de beneficios económicos por territorio.

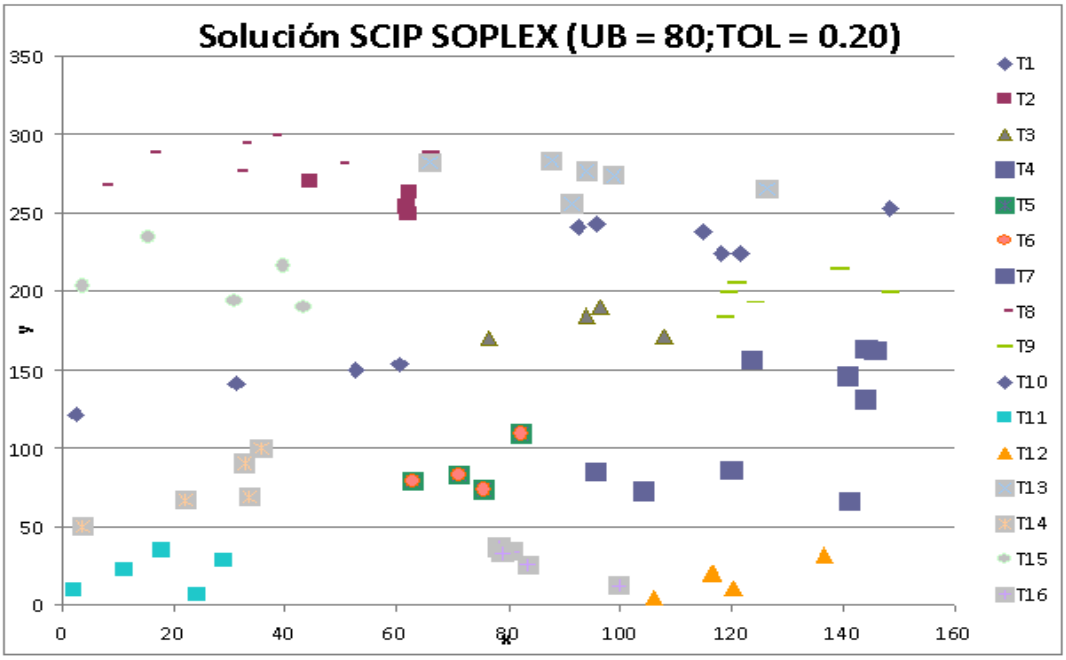


Figura 6.3 Ilustración de solución para 80 unidades básicas

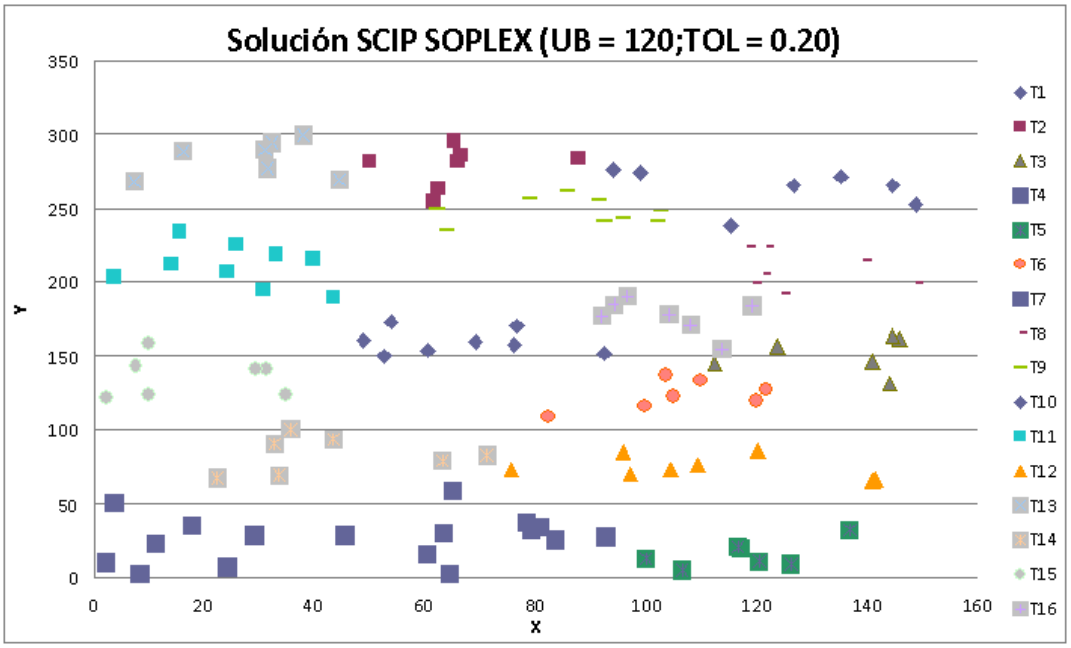


Figura 6.4 Ilustración de solución para 120 unidades básicas

Las figuras 6.3 y 6.4, nos permiten apreciar la forma de los territorios diseñados mediante el modelamiento matemático. Se observa que a pesar de no haber incluido la restricción de contiguidad, no se presentan casos de territorios no conexos, o separados por otro territorio. Esto se debe a que la función objetivo empleada, es una medida de compacidad de los territorios. Lo cual de alguna forma trata de suplir, la no presencia de las restricciones mencionadas en el modelo.

A continuación se examinarán las soluciones dadas por la **heurística geométrica**. La figura 6.5 presenta los valores correspondientes a los límites superior e inferior para el criterio de beneficios económicos por territorio, según su número de unidades básicas.

	Tolerancias											
	0.05		0.1		0.2		0.3		0.4		0.5	
# Unidades Básicas	Límite Inf.	Límite Sup.	Límite Inf.	Límite Sup.	Límite Inf.	Límite Sup.	Límite Inf.	Límite Sup.	Límite Inf.	Límite Sup.	Límite Inf.	Límite Sup.
80	307.2	339.5	291.0	355.7	258.7	388.0	226.3	420.3	194.0	452.7	161.7	485.0
120	413.7	457.2	391.9	479.0	348.3	522.5	304.8	566.1	261.3	609.6	217.7	653.1
200	587.9	649.8	557.0	680.8	495.1	742.6	433.2	804.5	371.3	866.4	309.4	928.3
300	767.1	847.8	726.7	888.2	646.0	968.9	565.2	1049.7	484.5	1130.4	403.7	1211.2
500	1045.9	1156.0	990.8	1211.0	880.7	1321.1	770.6	1431.2	660.6	1541.3	550.5	1651.4

Figura 6.5 Número de unidades básicas y límites de criterio “Beneficios por territorio”

# Unidades Básicas	Valor Función Objetivo					
	0.05	0.1	0.2	0.3	0.4	0.5
80	2242.47	2236.57	2236.57	2106.81	2106.81	2106.81
120	3005.92	3359.77	3005.92	3005.92	3005.92	3005.92
200	5384.81	5411.46	5322.55	5322.55	5322.55	5322.55
300	9208.26	8640.54	8640.54	8640.54	8640.54	8640.54
500	15391.82	15391.82	15391.82	15391.82	15391.82	15391.82

Figura 6.6 Valor de la solución según el número de unidades básicas

Si se comparan los valores de la función objetivo tanto para la solución del modelo matemático, tabla 6.1, como los de la tabla 6.6, observaremos la mejor calidad de la solución del modelo matemático. Pero así mismo es evidente, el mayor tiempo de ejecución que requiere la solución del solver.

# Unidades Básicas	Tolerancias						Desv. Std. Beneficios $	Promedio Beneficio Territorio
	0.05	0.1	0.2	0.3	0.4	0.5		
80	16:22	5:34	2:13	2:45	2:47	2:46	17.89	323.33
120	5:40	1:02	2:47	2:35	2:35	2:35	17.37	435.43
200	3:00	2:42	3:01	3:00	2:59	3:00	17.10	618.87
300	2:24	3:09	3:10	3:10	3:11	3:11	16.70	807.44
500	3:30	3:34	3:34	3:33	3:34	3:34	16.15	1100.93

Figura 6.7 Tiempos de ejecución por número de unidades básicas y tolerancias (mm:ss)

La tabla 6.7 presenta los tiempos de ejecución de la heurística geométrica. Se observa que el mayor tiempo de ejecución se da cuando existe un número pequeño de unidades básicas, con una tolerancia de 0.05 y con la mayor desviación estándar. Esto se debe a dos factores:

1. Incremento en el número de rotaciones al eje de coordenadas de las unidades básicas.
2. Mayor número de relajaciones a los límites de tolerancia.

Lo antes mencionado, implica un mayor número de análisis de las unidades básicas, y por ende un mayor tiempo de ejecución. Además observamos en la figura 6.7 que a medida que se aumenta la tolerancia, el tiempo de ejecución tiende a disminuir y a estabilizarse, ya que esto contribuye a eliminar el número de relajaciones de los límites del criterio de factibilidad.

Dicha figura nos muestra que a medida que se aumenta el número de unidades básicas el tiempo de ejecución tiende a estabilizarse, para un número fijo de territorios. Lo que ilustra el buen rendimiento de la heurística para conjuntos grandes de unidades básicas, tal situación se grafica en la figura 6.8. Esto ocurre debido a que debe examinar menos unidades básicas para hallar soluciones factibles.

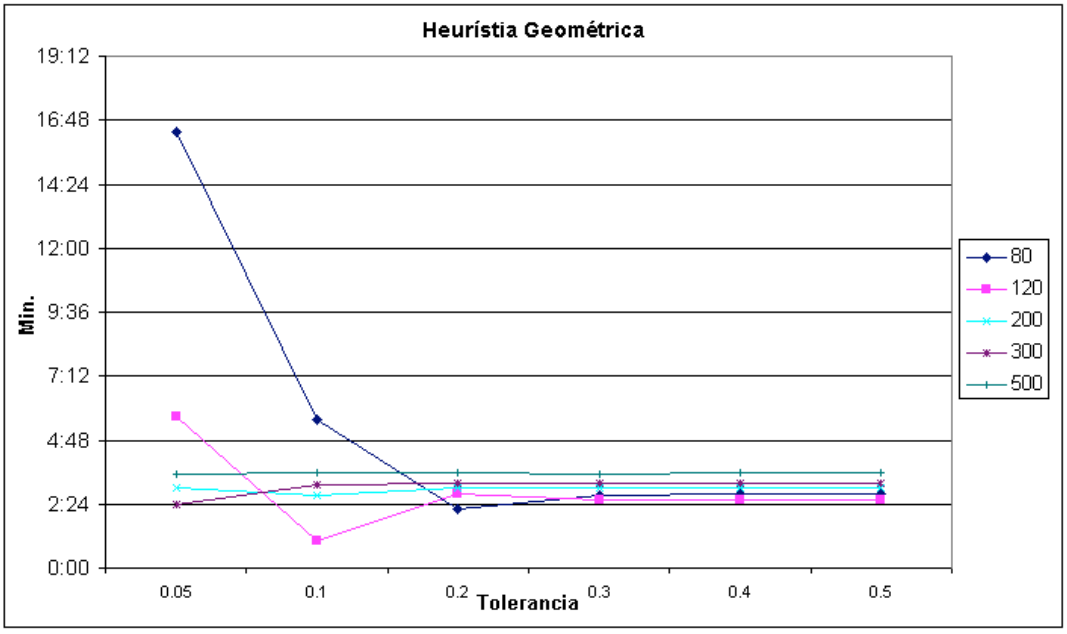


Figura 6.8 Tiempo de ejecución por tolerancia

En la figura 6.9 se presentan los valores del criterio de balance analizado en los territorios que forman parte de la solución, para el caso donde se procesaron 500 unidades básicas con una tolerancia de 0.05. Se observa la buena calidad de la solución respecto a los límites de tolerancia.

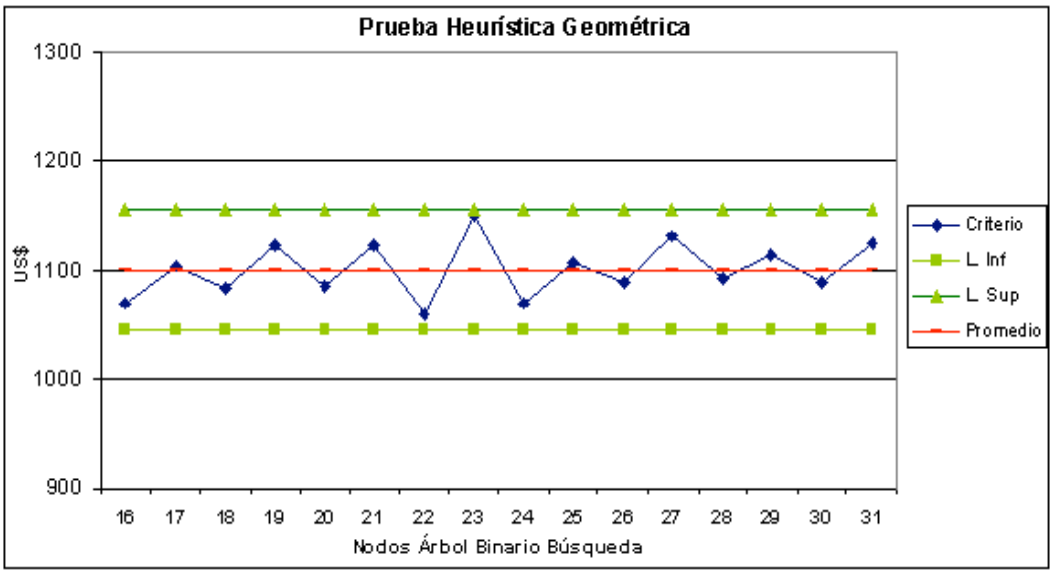


Figura 6.9 Solución Heurística (Unidades básicas = 500, Tolerancia = 0.05)

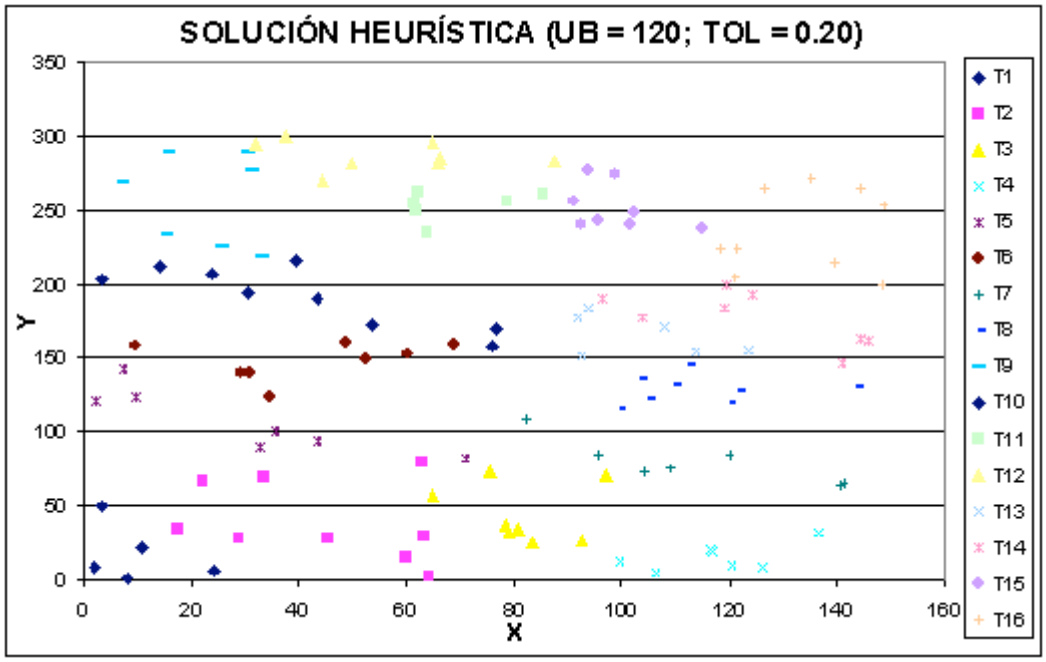


Figura 6.10 Ilustración de solución heurística para 120 unidades básicas

Cabe mencionar que los tiempos de ejecución de la heurística del TSP, fueron cercanos a un segundo.

CONCLUSIONES

1. Un diseño territorial balanceado respecto a beneficios económicos y carga de trabajo, proporciona un ambiente justo para la fuerza de ventas de una empresa, y un nivel de ingresos proporcional al esfuerzo del equipo de vendedores.

2. Entre las diversas alternativas para mejorar la productividad de la fuerza de ventas, el diseño de territorios comerciales constituye la de mayor impacto en el nivel de penetración de mercado.

3. La solución del problema del agente viajero dentro de cada territorio, permite brindar un excelente nivel de servicio a los clientes en términos de puntualidad en la entrega de los productos. Además contribuye a mejorar la rentabilidad de la operación.

4. Para el caso de 80 unidades la heurística geométrica halló una solución 1.8 veces superior a la solución óptima y para el caso de 120 unidades básicas fue de 1.6 veces aproximadamente

5. En los diferentes escenarios, se observó que la heurística geométrica presentó un tiempo de ejecución considerablemente menor a la solución del modelo matemático.

6. El tiempo de ejecución de la implementación en SCIP, creció de forma exponencial en función del número de unidades básicas a zonificar.

7. El tiempo de ejecución de la heurística geométrica tiende a estabilizarse a medida que aumenta el número de unidades básicas.

RECOMENDACIONES

1. En base a los beneficios de un diseño territorial balanceado y a una secuencia eficiente de visita, se recomienda a las empresas de consumo masivo que zonifiquen una ciudad según ciertos criterios de interés con la finalidad de incrementar la productividad del equipo de vendedores.

2. Incorporar al proceso de toma de decisiones en los diferentes niveles de una organización, el uso de técnicas de investigación de operaciones.

3. Fomentar una cultura de recopilación de datos inherentes a las operaciones diarias de distribución, con la finalidad de disponer de datos confiables para los modelos matemáticos y métodos heurísticos.

4. Para disminuir la complejidad de la región factible, se recomienda un procesamiento previo de las unidades básicas del problema original. Tal procesamiento consiste en agrupar las unidades básicas que por su naturaleza deban pertenecer a un mismo territorio. Dichos conjuntos de unidades básicas se tratarán como una sola unidad básica, cuyas medidas de actividad serán igual a la suma de las medidas de las actividades de las unidades básicas que las conforman.

Bibliografía

[1] David G. Luenberger, Yinyu Ye, INTERNATIONAL SERIES IN PERATIONS RESEARCH&MANAGEMENT SCIENCE, Frederick Series Editor, Universidad Stanford. Linear and nonlinear Programming third Edition, Pags. 515-521.

[2] David G. Luenberger, Yinyu Ye, INTERNATIONAL SERIES IN PERATIONS RESEARCH&MANAGEMENT SCIENCE, Frederick Series Editor, Universidad Stanford. Linear and nonlinear Programming third Edition, Pags. 19-24.

[3] Enrique Castillo, Antonio J. Conejo, Pablo Pedregal, Ricardo García y Natalia Alguacil, 20 de febrero de 2002, Formulación y Resolución de Modelos de Programación Matemática en Ingeniería y Ciencia. Pag. 119.

[4] H. Paul Williams, Model Building in Mathematical Programming, 4th Edition, Department of Operational Research, London School of Economics, 2008, pag. 151.

[5] http://wapedia.mobi/es/Grafo_simple.

[6] Ahuja, Magnanti, Orlin, Network Flows Theory Algorithms and Applications, Prentice Hall, 1993, Pags. 23-31.

[7] Flores Rivas, Ríos Mercado, Estudio computacional sobre un problema de división de territorios comerciales, Universidad del Valle de México, 2009, pag. 43.

[8] Segura, Mercado, A location-Allocation heuristic for a territory design problem in beverage distribution firm, International Conference on Industrial Engineering, Theory, Applications and Practice, Cancun, Mexico, 2007

[9] Kalcsics J., Nickel S. and Schroder, Towards a Unified Territory Design Approach – Applications, Algorithms and GIS Integration, (2005), Pag. 24 – 32.

[10] Phan Thanh An Institute of Mathematics Vietnamese Academy of Science and Technology, A modification of Graham's algorithm for determining the convex hull of a finite planar set, Pag. 4.

[11] Alfredo Olivera: Heurísticas para Problemas de Ruteo de Vehículos, Instituto de Computación, Facultad de Ingeniería, Universidad de la República, Montevideo, Uruguay, 2004, pag. 1.

[12] Dantzig, G., Fulkerson, D., Johnson, S.: Solution of a large scale travelling salesman problem. Operations Research 2, 1954, 393–410.

[13] Johnson, McGeoch, Rothberg: Asymptotic experimental analysis for the Held-Karp traveling salesman bound, Atlanta, Georgia, United States, 1996, pag.: 341 – 350.

[14] Cook, Cunningham, Pulleyblank, Shrijver: Combinatorial Optimization, A. Wiley, Interscience Publication, 1998, pag. 243.

[15] Rozenkrantz, Stearns, y Lewis: An Analysis of Several Heuristics for the Traveling Salesman Problem, SIAM J. Comput., 1977, pag.: 563-581.

[16] Tobias Achterberg, Constraint Integer Programming, Ph.D. thesis, TU Berlin, July 2007.

[17] http://scip.zib.de/whatis.shtml

[18] http://scip.zib.de/

[19] **Wunderling, Roland**: Paralleler und Objektorientierter Simplex- Algorithmus, PhD thesis, Technische Universit at Berlin, 1996

[20] http://www.wm.uni-bayreuth.de/index.php?id=434&L=3