INSTITUTO TECNOLÓGICO DE CD. MADERO

DIVISIÓN DE ESTUDIOS DE POSGRADO E INVESTIGACIÓN

"BÚSQUEDA TABÚ APLICADA AL PROBLEMA ROBUSTO DE ABASTECIMIENTO INTERNACIONAL CON CAPACIDAD FINITA (ROCIS)"

PARA OBTENER EL GRADO DE:

MAESTRO EN CIENCIAS
EN CIENCIAS DE LA COMPUTACIÓN

P R E S E N T A:

I.S.C. SANTIAGO GÓMEZ CARPIZO

D I R E C T O R:
DR. HÉCTOR JOAQUÍN FRAIRE HUACUJA

C O D I R E C T O R:
DR. JUAN JAVIER GONZÁLEZ BARBOSA

CD. MADERO, TAM. MÉXICO OCTUBRE 2007

CONTENIDO

ÍNDICE DE CONTENIDO …… i
ÍNDICE DE FIGURAS ……... iv
ÍNDICE DE TABLAS ……. vi

Capítulo 1. INTRODUCCIÓN 1
1.1 Descripción del problema de investigación ……..... 2
1.1.1 Definiciones básicas ……. 2
1.1.2 Subproblema de transporte asociado a cada escenario ……. 2
1.1.3 Función objetivo del problema…….. 3
1.2 Objetivo general …….. 3
1.2.1 Objetivos específicos……. 3
1.3 Justificación ……. 4
1.4 Alcances y Limitaciones……... 4
1.5 Organización del documento…… 5

Capítulo 2. MARCO TEÓRICO 6
2.1 Programación lineal (PL)……. 7
2.1.1 Áreas de aplicación…… 7
2.1.2 Casos especiales de programación lineal: Problema de transporte… 8
2.1.2.1 Ejemplo clásico del problema de transporte……. 8
2.1.2.2 Solución automatizada del ejemplo clásico del problema de transporte …….. 10
2.1.2.3 Elementos asociados a la programación lineal ……. 12
2.2 Matrices Dispersas…… 19

2.2.1 Representación de matrices dispersas mediante 3 vectores…… 20
2.2.2 Procedimiento de representación de matrices dispersas mediante 3 vectores…….. 20
2.3 Interfase de programación de LINDO 2.0 ..…… 21
2.3.1 Representación en la matriz de restricciones ……... 22
2.3.2 Estructuras de datos utilizadas por el API de LINDO……... 22
2.3.3 Codificación en lenguaje C……. 24
2.4 Trabajos relacionados……... 27

Capítulo 3. SOLUCIÓN DE REFERENCIA 32
3.1 Descripción general…….. 33
3.2 Representación binaria y decimal de las soluciones……. 36
3.3 Estructuras de datos utilizadas…….. 37
3.4 Funciones utilizadas…….. 43
3.4.1 GeneraciónVoraz()…… 43
3.4.1.1 Función Initialize()……... 44
3.4.1.2 Función select_min()…… 45
3.4.2 Función BúsquedaLocal()…….. 46
3.4.2.1 Función form_list()……...…….. 48
3.4.2.2 Función select_move()……. 55
3.4.2.3 Función execute_move()……. 61

Capítulo 4. PROPUESTA DE SOLUCIÓN 63
4.1 Enfoque de solución……. 64
4.2 Estrategias de prioridades consideradas……... 64
4.2.1 Estrategia E_1: proveedores de menor costo fijo y mayor capacidad de producción…… 65

4.2.2 Proveedores de menor costo fijo, mayor capacidad de producción y menor valor esperado del costo de enviar a todas las plantas (E_2). 66
4.2.3 Proveedores de menor costo fijo, mayor capacidad de producción y menor valor esperado del costo de enviar a las plantas frecuentemente servidas, refinando la solución inicial previa a la búsqueda local (E_3)…… 66
4.2.4 Proveedores de menor costo fijo, mayor capacidad de producción y menor valor esperado del costo de enviar a las plantas de menor costo (E_4 y E_5)…… 69

Capítulo 5. RESULTADOS EXPERIMENTALES 73
5.1 Plataforma experimental…… 74
5.2 Experimento. Análisis comparativo del desempeño de las estrategias propuestas…….. 75
5.2.1 Objetivo…… 75
5.2.2 Procedimiento……... 75
5.2.3 Resultados…… 75
5.2.4 Análisis de resultados…….. 77
5.3 Experimento 2. Análisis de la eficiencia de la estrategia de referencia respecto a la estrategia E_5 con alfa en 0.8 …… 78
5.3.1 Objetivo ……... 78
5.3.2 Procedimiento……... 78
5.3.3 Resultados…… 78
5.3.4 Análisis de resultados……... 79
Capitulo 6. CONCLUSIONES Y TRABAJOS FUTUROS 80
6.1 Aportaciones…… 81
6.2 Trabajos futuros……... 82
REFERENCIAS …… 83

ÍNDICE DE FIGURAS

Figura 2.1 Capacidad de producción de las plantas de la empresa …… 8

Figura 2.2 Demanda los centros de distribución de la empresa ……. 9

Figura 2.3 Costos unitarios de transporte …….. 9

Figura 2.4 Representación del problema ……... 10

Figura 2.5 Representación del problema ……... 11

Figura 2.6 Script proporcionado a LINDO …… 11

Figura 2.7 Salida de LINDO ……. 11

Figura 2.8 Unidades que minimizan la función objetivo y satisfacen las restricciones…… 12

Figura 2.9 Resultados de costos reducidos del ejemplo resuelto ……... 13

Figura 2.10 Costo reducido de la variable x_{14} del ejemplo resuelto …… 13

Figura 2.11 Relación de variables de holgura …….. 13

Figura 2.12 Variables de holgura de la restricción 3……. 14

Figura 2.13 Relación de precios sombra……... 14

Figura 2.14 Comportamiento del uso de los precios sombra ……... 15

Figura 2.15 Precio sombra de la restricción 3 …….. 15

Figura 2.16 Relación de análisis de sensibilidad de los coeficientes …….. 16

Figura 2.17 Análisis de sensibilidad del coeficiente de la variable x_{12}…… 17

Figura 2.18 Valores de Salida…… 17

Figura 2.19 Relación de análisis de sensibilidad de los recursos disponibles... 18

Figura 2.20 Análisis de sensibilidad de la restricción 2…… 18

Figura 2.21 Valores de Salida…… 19

Figura 2.22 Ejemplo de matriz dispersa…… 20

Figura 2.23 Ejemplo de un problema de programación lineal…….. 21

Figura 2.24 Matriz de restricciones del problema considerado……. 22

Figura 2.25 Código fuente del problema considerado……... 24

Figura 3.1 Algoritmo de la solución referencia……... 33

Figura 3.2 Algoritmo de la función Initialice()……... 44

Figura 3.3 Algoritmo de la función select_min()…… 46
Figura 3.4 Algoritmo de la función BusquedaLocal ()…… 48
Figura 3.5 Algoritmo de la función form_list ()…….. 52
Figura 3.6 Algoritmo select_move…….. 58
Figura 3.7 Algoritmo execute_move……... 61

ÍNDICE DE TABLAS

Tabla 2.1	Estructuras de datos utilizadas por la interfaz de LINDO ……	23
Tabla 2.2	Resumen de artículos que tratan el problema de la ubicación de la planta…….	29
Tabla 2.3	Características revelantes……..……....	31
Tabla 3.1	Estructuras utilizadas por la interfaz de LINDO……...	37
Tabla 3.2	Estructura sInterface: datos de entrada para el proceso de optimización……..	38
Tabla 3.3	Estructura sInterface: información de salida del proceso de optimización que se genera al invocar la función de LINDO GetInfo()……	38
Tabla 3.4	Estructura sInterfaceComunes: datos de entrada al proceso de optimización que no son dependientes de la solución analizada o del escenario……..	39
Tabla 3.5	Estructura inputdata……...	40
Tabla 3.6	Estructura soldata: información de la solución o selección de proveedores analizada que se genera en el proceso de solución...	40
Tabla 3.7	Estructura movedata: información relativa al movimiento (inserción, eliminación o intercambio de proveedores) realizado para construir una solución actual…….	41
Tabla 3.8	Funciones utilizadas para resolver el problema ROCIS……	43
Tabla 3.9	Funciones utilizadas por la función GeneraciónVorazl()……..	44
Tabla 3.10	Listas globales que permiten mantener un registro histórico de las soluciones evaluadas en el proceso……..	47
Tabla 3.11	Conceptos utilizados en la función BúsquedaLocal()……...	47
Tabla 3.12	Listas de movimientos de inserción, eliminación e intercambio...	49
Tabla 3.13	Listas tabú de inserción, eliminación e intercambio…….	49
Tabla 4.1	Construcción de la solución inicial con la estrategia E_1……	65

Tabla 4.2 Construcción de solución inicial con la estrategia E_2…… 66
Tabla 4.3 Construcción de solución inicial con la estrategia E_3…… 67
Tabla 4.4 Construcción de solución inicial con la estrategia E_4…… 70
Tabla 4.5 Construcción de solución inicial con la estrategia E_5…… 71
Tabla 5.1 Resultados de referencia…… 75
Tabla 5.2 Resultados que se obtienen al utilizar la estrategia E_2……... 76
Tabla 5.3 Resultados que se obtienen al utilizar la estrategia E_3……... 76
Tabla 5.4 Resultados que se obtienen al utilizar la estrategia E_4……... 76
Tabla 5.5 Resultados que se obtienen al utilizar la estrategia E_5 con α=0.8 77
Tabla 5.6 Resultados comparativos de las estrategias propuestas……. 78
Tabla 5.7 Resultados que se obtienen al utilizar la solución de referencia con 10 iteraciones…….. 79
Tabla 5.8 Resultados que se obtienen al utilizar la estrategia E_5 con α=0.8 con 10 iteraciones……. 79

ÍNDICE DE FIGURAS

Figura 2.1 Capacidad de producción de las plantas de la empresa 8

Figura 2.2 Demanda los centros de distribución de la empresa 9

Figura 2.3 Costos unitarios de transporte 9

Figura 2.5 Representación del problema como la relación entre cantidad a enviar, demanda y capacidad 10

Figura 2.6 Script requerido por LINDO para resolver el problema 11

Figura 2.7 Salida de LINDO 11

Figura 2.8 Unidades que minimizan la función objetivo y satisfacen las restricciones 12

Figura 2.9 Relación de costos reducidos del ejemplo resuelto 12

Figura 2.10 Costo reducido de la variable del ejemplo resuelto 13

Figura 2.11 Relación de variables de holgura 13

Figura 2.12 Variables de holgura de la restricción 3 14

Figura 2.13 Relación de precios sombra 14

Figura 2.14 Comportamiento del uso de los precios sombra 15

Figura 2.15 Precio sombra de la restricción 3 15

Figura 2.16 Relación de análisis de sensibilidad de los coeficientes 16

Figura 2.17 Análisis de sensibilidad del coeficiente de la variable 16

Figura 2.18 Comprobación del análisis de sensibilidad de coeficientes, variable 17

Figura 2.19 Relación de análisis de sensibilidad de los recursos disponibles 18

Figura 2.20 Análisis de sensibilidad de la restricción 2 18

Figura 2.21 Comprobación del análisis de sensibilidad de las restricciones, afectando la restricción 2 19

Figura 2.22 Ejemplo de matriz dispersa 20

Figura 2.23 Ejemplo de un problema de programación lineal 21

Figura 2.24 Matriz de restricciones del problema considerado 22

Figura 2.25 Código fuente del problema considerado 24
Figura 2.25 Continuación … 26
Figura 3.1 Algoritmo de la solución referencia 35
Figura 3.2 Algoritmo de la función Initialice() 44
Figura 3.3 Algoritmo de la función select_min() 46
Figura 3.4 Algoritmo de la función BusquedaLocal () 48
Figura 3.5 Algoritmo de la función form_list () 54
Figura 3.6 Algoritmo select_move 60
Figura 3.7 Algoritmo execute_move 62

ÍNDICE DE TABLAS

Tabla 2.1 Estructuras de datos utilizadas por la interfaz de LINDO 22
Tabla 2.2 Resumen de artículos que tratan el problema de la ubicación de la planta. 29
Tabla 2.3 Estrategias de selección de proveedores utilizadas en los métodos de solución del problema de abastecimiento internacional con capacidad finita reportados en la literatura. 30
Tabla 3.1 Estructuras utilizadas por la interfaz de LINDO 37
Tabla 3.2 Estructura sInterface: datos de entrada para el proceso de optimización 37
Tabla 3.3 Estructura sInterface: información de salida del proceso de optimización que se genera al invocar la función de LINDO GetInfo(). 38
Tabla 3.4 Estructura sInterfaceComunes: datos de entrada al proceso de optimización que no son dependientes de la solución analizada o del escenario. 39
Tabla 3.5 Estructura inputdata 39
Tabla 3.6 Estructura soldata: información de la solución o selección de proveedores analizada que se genera en el proceso de solución 40
Tabla 3.7 Estructura movedata: información relativa al movimiento (inserción, eliminación o intercambio de proveedores) realizado para construir una solución actual. 41
Tabla 3.8 Funciones utilizadas para resolver el problema ROCIS. 42
Tabla 3.9. Funciones utilizadas por la función GeneraciónVorazl(). 43
Tabla 3.10 Listas globales que permiten mantener un registro histórico de las soluciones evaluadas en el proceso. 46
Tabla 3.11 Conceptos utilizados en la función BúsquedaLocal() 46
Tabla 3.12 Listas de movimientos de inserción, eliminación e intercambio. 48
Tabla 3.13 Listas tabú de inserción, eliminación e intercambio. 49
Tabla 4.1 Construcción de la solución inicial con la estrategia E1 66

Tabla 4.2 Construcción de solución inicial con la estrategia E2 67
Tabla 4.3 Construcción de solución inicial con la estrategia E3 68
Tabla 4.4 Construcción de solución inicial con la estrategia E4 71
Tabla 4.5 Construcción de solución inicial con la estrategia E5 72
Tabla 5.1 Resultados de referencia 76
Tabla 5.2 Resultados que se obtienen al utilizar la estrategia E2 77
Tabla 5.3 Resultados que se obtienen al utilizar la estrategia E3 77
Tabla 5.4 Resultados que se obtienen al utilizar la estrategia E4 77
Tabla 5.5 Resultados que se obtienen al utilizar la estrategia E5 con □=0.8 78
Tabla 5.6 Resultados comparativos de las estrategias propuestas 79

DEDICATORIA

A Dios, por darme la vida y llenarme de bendiciones.

A mi esposa, por ser mi razón de ser, mi puerto en mi oscuridad, por alentarme a seguir avanzando en mi sed de superación, por ser mi amiga y confidente. Esposa mía, eterna novia mía.

A mi padre, por enseñarme a luchar por mis ideales, sin dejar de ayudar a los demás en el andar.

A mi madre, por guiarme a ser un buen hombre, por darme su apoyo incondicional, por llorar mis tropiezos y reír en mis logros. Por ser mi amiga y consejera.

A mi tía Leonor, mi segunda madre. Por estar conmigo en cada momento de mi vida, siempre sonriendo y alentándome a seguir avanzando. Gracias tilliya.

A mi hermano, por hacerme ver, que realmente la hacia en sistemas más que en eléctrica.

A Don Mario Gavito, que me enseño el verdadero valor de ser amigo.

Al Dr. Apolinar, por tratar de guiarme en mi carrera.

A mis suegros, por las porras y buenas vibras.

A Don Alvaro y su familia, gracias por su amistad.

AGRADECIMIENTOS

Le doy las gracias al Dr. Héctor Joaquín Fraire Huacuja, quien es el asesor de este proyecto, no solo por sus valiosos consejos, si no por darme lecciones de vida. Gracias maestro, por convertir mis tropiezos en avances de mi formación.

Agradezco, también a los profesores que forman parte del comité de revisión de esta tesis: Dr. Juan Javier Gonzáles Barbosa, Dr. José Antonio Martines Flores, Dr. Arturo Hernández Ramírez y el Dr. Héctor Joaquín Fraire Huacuja, sus observaciones y enseñanzas han sido los pilares que sustentan la realización de esta tesis.

Gracias a los profesores de la maestría en ciencias en ciencias de la computación, por compartir sus conocimientos y valores necesarios para concretar mis metas.

Agradezco también, al Consejo Nacional de Ciencia y Tecnología (CONACYT), por haber provisto los recursos económicos para el desarrollo de esta investigación, y para mi formación como investigador.

Finalmente, agradezco profundamente al Instituto Tecnológico de Ciudad Madero (ITCM), por darme la oportunidad de continuar mis estudios.

De paso, a mis compañeros de la maestría: Tania, Vero, Bárbara, Diana, Oscar, Pedro, Francisco, Margarito y Miguel.

Santiago Gómez Carpizo.

RESUMEN

El problema robusto del abastecimiento internacional con capacidad finita (ROCIS), consiste en seleccionar un conjunto de proveedores, para satisfacer la demanda de productos de un conjunto de plantas localizadas en diferentes países. En esta tesis, se analizan diferentes estrategias de prioridades para la generación de soluciones iniciales de la búsqueda tabú del problema ROCIS. Las cuales, se utilizan para elegir los proveedores que se incorporan a una solución inicial. La primera, consiste en dar prioridad a los proveedores de menor costo fijo y mayor capacidad de producción, mientras que la segunda, incorpora el valor esperado del costo de enviar productos del proveedor a todas las plantas. Una limitación de la primera alternativa es que no considera el costo de envío y aún cuando este factor sí es considerado por la segunda, el mecanismo que se utiliza resulta demasiado pesimista. En este trabajo, se propone modificar el mecanismo de incorporación del costo de envío, para que se consideren únicamente las plantas hacia las que resulta más económico el envío de los productos desde el sitio del proveedor. Para validar lo anterior, se proponen dos modelos alternativos. Los resultados experimentales muestran que una de las estrategias propuestas logra reducir en un 3.32% el consumo de recursos requeridos para resolver las instancias y en un 21.05% los recursos requeridos para llegar a la mejor solución. Además de esta reducción en el consumo de recursos, se logra incrementar la calidad de la solución en un 1.14%. Dado lo alentador de los resultados, actualmente se esta trabajando en la aplicación de estas estrategias en la mejora del desempeño de la solución de ROCIS que utiliza reencadenamiento de trayectorias.

Capítulo 1

INTRODUCCIÓN

En este capítulo se presenta un panorama general de la tesis, el cual inicia con la descripción de los motivos que llevaron al desarrollo de esta investigación y continúa con la definición del problema que se aborda en este estudio. Se presentan los objetivos que se plantea alcanzar y el contexto de dicho estudio. Se hace una breve introducción a los elementos de la programación lineal y se termina con una descripción del contenido de cada capítulo de este documento.

ÍNDICE DE CONTENIDO

ÍNDICE DE CONTENIDO i
ÍNDICE DE FIGURAS i
ÍNDICE DE TABLAS i
1.1 Descripción del problema de investigación 2
1.1.1 Definiciones básicas 2
1.1.2 Subproblema de transporte asociado a cada escenario 3
1.2 Objetivo general 3
1.2.1 Objetivos específicos 3
1.3 Justificación 4
1.4 Alcances y Limitaciones 4
1.5 Organización del documento 5

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

1.1 Descripción del problema de investigación

El problema robusto del abastecimiento internacional con capacidad finita (ROCIS) consiste en seleccionar un conjunto de proveedores para satisfacer la demanda de productos de un conjunto de plantas localizadas en diferentes países. En el modelo se considera un solo producto en un solo periodo. La incertidumbre de la demanda y la tasa de cambio se modelan utilizando un conjunto de escenarios.

1.1.1 Definiciones básicas

En el modelo se utilizan las siguientes definiciones:

Parámetros:

N : Conjunto de plantas internacionales $\{1,2,..,n\}$

M : Conjunto de proveedores internacionales $\{1,2,..,m\}$

S : Conjunto de escenarios

f_i : Costo fijo asociado al proveedor i

c_{ij} : Costo unitario del envío de producto del proveedor i a la planta j

b_i : Capacidad del proveedor i

d_{ij} : Demanda de producto de la planta j en el escenario s

e_{is} : Tipo de cambio en la localidad del proveedor i en el escenario s

p_s : Probabilidad de ocurrencia del escenario s

Variables:

x_{ijs} : Cantidad de producto enviada del proveedor i a la planta j en el escenario s

y_i : es 1 si el proveedor i es seleccionado y 0 en caso contrario

1.1.2 Subproblema de transporte asociado a cada escenario

Para cada elección de proveedores y_i donde $i = 1, 2, .., m$ se resuelve el problema de distribución asociado con cada escenario y se determina su costo mínimo z_s

Minimizar

$$z_s = \sum_{i \in M} \sum_{j \in N} e_{is} c_{ij} x_{ijs} \tag{1.1}$$

Sujeto a:

$$\begin{aligned} &\sum_{i \in M} x_{ijs} \geq d_{js} && \forall j \in N \\ &\sum_{i \in N} x_{ijs} \leq b_i y_i && \forall i \in M \\ &x_{ijs} \geq 0 && \forall i \in M,\ j \in N \end{aligned} \tag{1.2}$$

1.1.3 Función objetivo del problema

Minimizar

$$F(y) = \sum_{s \in S} p_s \left(\sum_{i \in M} e_{is} f_i y_i + z_s \right) + \mu \sqrt{\frac{\sum_{s \in S^+} p_s \left(z_s - E(z_s) \right)^2}{\sum_{s \in S^+} p_s}} \tag{1.3}$$

donde

$$S^+ = \left\{ s : z_s - E(z_s) \geq 0 \right\}$$

$$E(z_s) = \sum_{s \in S} p_s z_s$$

1.2 Objetivo general

Diseñar e implementar soluciones competitivas del problema ROCIS utilizando diferentes configuraciones de los parámetros de búsqueda tabú.

1.2.1 Objetivos específicos

- Análisis e implementación de la solución tabú
- Análisis e implementación de diferentes estrategias de prioridades
- Evaluación experimental del desempeño de los algoritmos

1.3 Justificación

El desarrollo de esta investigación analiza la variante propuesta por González-Velarde y Laguna [González 2004], la cual considera un solo producto en un solo periodo, y modela la incertidumbre de la demanda, así como la tasa de cambio mediante un conjunto de escenarios.

En la formulación del presente problema, los costos dependen de las condiciones económicas de los países en que se localizan los proveedores y las plantas. Además, se considera que son inciertas las demandas de las plantas y la tasa de cambio de los países donde se ubican los proveedores y que la capacidad de producción de los proveedores es finita.

La formulación robusta considera que una solución es factible sí y solo sí es factible en todos los escenarios. La función objetivo utilizada minimiza el valor esperado del costo y penaliza a las soluciones cuyo costo óptimo en algún escenario supera el valor esperado de los costos óptimos en todos los escenarios. A través de este mecanismo se incorpora el riesgo vinculado a la incertidumbre representada mediante el conjunto de escenarios considerados.

1.4 Alcances y Limitaciones

- El código del proyecto, se desarrolló en C estándar.
- Sólo se consideran, las instancias reportadas en [González 2004] para la evaluación de los algoritmos de solución
- El optimizador lineal de API LINDO versión 2.0, es utilizado para resolver los modelos de programación lineal.
- La solución tabú de referencia, es presentada en [González 2004].

1.5 Organización del documento

En el capitulo 2, se fundamenta el desarrollo de los diferentes elementos de esta tesis. En el capitulo 3, se presenta la solución de referencia. En el capitulo 4, se presenta el enfoque de solución, además, se describen las estrategias propuestas. En el capitulo 5, se describen los experimentos realizados y el análisis de resultados para determinar la calidad y la eficiencia de las estrategias propuestas. Finalmente, en el capitulo 6, se describen las conclusiones de esta tesis y las líneas de investigación que se identificaron en el proceso.

Capítulo 2

MARCO TEÓRICO

En este capítulo se presentan los fundamentos teóricos que sustentan esta investigación. Se describe el desarrollo de la temática de la programación lineal, se menciona un caso tipo del problema de transporte, del uso de las matrices dispersas, de su solución computacional usando el optimizador LINDO y el lenguaje C. Finalmente se presenta el análisis de los trabajos relacionados más relevantes que abordan algunos aspectos del problema ROCIS.

CONTENIDO

CONTENIDO 7
FIGURAS 7
TABLAS 8
2.1 Programación lineal (PL) 7
2.1.1 Áreas de aplicación: 7
2.1.2 Casos especiales de programación lineal: Problema de transporte 8
2.2 Matrices Dispersas 19
2.2.1 Representación de matrices dispersas mediante 3 vectores. 20
2.2.2 Procedimiento de representación de matrices dispersas mediante 3 vectores 20
2.3 Interfase de programación de LINDO 2.0 21
2.3.1 Representación en la matriz de restricciones 22
2.3.2 Estructuras de datos utilizadas por el API de LINDO 22
2.3.3 Codificación en lenguaje C. 24
2.4 Trabajos relacionados 27

FIGURAS

Figura 2.1 Capacidad de producción de las plantas de la empresa 8
Figura 2.2 Demanda los centros de distribución de la empresa 9
Figura 2.3 Costos unitarios de transporte 9
Figura 2.5 Representación del problema 11
Figura 2.6 Script proporcionado a LINDO. 11
Figura 2.7 Salida de LINDO 11
Figura 2.8 Unidades que minimizan la función objetivo 12
Figura 2.9 Resultados de costos reducidos del ejemplo resuelto 13
Figura 2.10 Costo reducido de la variable x_{14} del ejemplo resuelto 13
Figura 2.11 Relación de variables de holgura 13
Figura 2.12 Variables de holgura de la restricción 3 14
Figura 2.13 Relación de precios sombra 14
Figura 2.14 Comportamiento del uso de los precios sombra 15
Figura 2.15 Precio sombra de la restricción 3 15
Figura 2.16 Relación de análisis de sensibilidad de los coeficientes 16
Figura 2.17 Análisis de sensibilidad del coeficiente de la variable x_{12} 17
Figura 2.18 Valores de Salida 17
Figura 2.19 Relación de análisis de sensibilidad de los recursos disponibles 18
Figura 2.20 Análisis de sensibilidad de la restricción 2 18
Figura 2.21 Valores de Salida 19
Figura 2.22 Ejemplo de matriz dispersa 20
Figura 2.23 Ejemplo de un problema de programación lineal 21
Figura 2.24 Matriz de restricciones del problema considerado 22
Figura 2.25 Código fuente del problema considerado 24
Figura 2.25 Continuación …. 25
Figura 2.25 Continuación …. 26

TABLAS

Tabla 2.1 Estructuras de datos utilizadas por la interfaz de LINDO..... 23
Tabla 2.2 Resumen de artículos que tratan el problema de la ubicación de la planta..... 29
Tabla 2.3 Características relevantes..... 31

2.1 Programación lineal (PL)

Un problema de programación lineal es un caso simplificado del problema general de optimización matemática. Consiste en una función objetivo lineal y una o más restricciones lineales [LINDO].

Fue desarrollado para resolver problemas de planeación de la fuerza aérea americana. La programación lineal permite seleccionar los mejores valores de las variables de decisión en situaciones donde las variables compiten con recursos limitados. Un problema de programación lineal se define como:

$$\begin{aligned} &Maximize / Minimize \quad Z = \sum_{j=1}^{n} c_j x_j \\ &Sujeto \quad a: \\ &\sum_{j=1}^{n} a_{ij} x_j \quad \{\leq, \equiv, \geq\} \quad b_i \quad (i = 1, ..., m) \\ &x_j \geq 0 \quad (j = 1, ...n) \end{aligned} \tag{2.1}$$

Donde:

- *n* Número de variables de decisión.
- *m* Número de restricciones.
- *a* Coeficiente conocido de cada variable en las restricciones.
- *b* Valor conocido ubicado en el lado derecho de las restricciones.
- *c* Coeficiente conocido que denota la relación costo/beneficio de la variable de decisión.

2.1.1 Áreas de aplicación:

- Desarrollar calendarios de trabajo que utilicen eficientemente el tiempo del personal.
- Asignar espacios de fabricación que permitan mejorar las líneas de producción.

- Determinar los niveles de gastos de varias actividades.
- Optimizar la mezcla de productos de una operación.
- Asignación del equipo para los vehículos del sistema de transporte publico.
- Problemas clásicos de transporte
    - Problema del cartero
    - Problema del agente viajero
    - Problema de rutas de vehículos
    - Problema del transporte

2.1.2 Casos especiales de programación lineal: Problema de transporte

Este tipo de problemas consisten en minimizar el costo de transporte de unidades desde cierto número de lugares de origen a cierto número de lugares de destino [Anderson 1991].

2.1.2.1 Ejemplo clásico del problema de transporte

Una empresa enfrenta el siguiente problema de transporte: minimizar el costo de transporte de productos desde 3 plantas (P1, P2 y P3) hacia 4 centros de distribución (D1, D2, D3 y D4).

En la Figura 2.1, se muestra la capacidad de producción de cada planta.

Origen	Planta	Capacidad de Producción (unidades)
1	P1	5,000
2	P2	6,000
3	P3	2,500
Total		13,500

Figura 2.1 Capacidad de producción de las plantas de la empresa

En la Figura 2.2, se muestra las demandas respectivas de los centros de distribución:

Destino	Centro de distribución	Demanda (unidades)
1	D1	6,000
2	D2	4,000
3	D3	2,000
4	D4	1,500
Total		13,500

Figura 2.2 Demanda los centros de distribución de la empresa

El objetivo es determinar las rutas que deben utilizarse y la cantidad que se debe de enviar a través de cada ruta, de tal manera que el costo total de transporte sea el mínimo y que satisfagan las demandas de todos los centros de distribución. En la Figura 2.3, se presentan los costos de transporte por unidad de producto:

		Destino			
		D1	**D2**	**D3**	**D4**
Origen	**P1**	3	2	7	6
	P2	7	5	2	3
	P3	2	5	4	5

Figura 2.3 Costos unitarios de transporte

Considerando el planteamiento anterior, se describen las etapas de la representación del problema de transporte:

- **Solución**

Se considera x_{ijs} como el número de unidades que serán transportadas desde la planta i al centro de distribución j.

La función objetivo es:

$$Min \quad (3x_{11} + 2x_{12} + 7x_{13} + 6x_{14}) + (7x_{21} + 5x_{22} + 2x_{23} + 3x_{24}) + (2x_{31} + 5x_{32} + 4x_{33} + 5x_{34}) \tag{2.2}$$

- Restricciones

1.- No exceder la capacidad de las plantas:

$$\begin{aligned} x_{11} + x_{12} + x_{13} + x_{14} &\le 5,000 \quad .....(1) \\ x_{21} + x_{22} + x_{23} + x_{24} &\le 6,000 \quad .....(2) \\ x_{31} + x_{32} + x_{33} + x_{34} &\le 2,500 \quad .....(3) \end{aligned} \tag{2.3}$$

2.- Satisfacer la demanda de los centros de distribución:

$$\begin{aligned} x_{11} + x_{21} + x_{31} &\ge 6,000 \quad .....(4) \\ x_{12} + x_{22} + x_{32} &\ge 4,000 \quad .....(5) \\ x_{13} + x_{23} + x_{33} &\ge 2,000 \quad .....(6) \\ x_{14} + x_{24} + x_{34} &\ge 1,500 \quad .....(7) \end{aligned} \tag{2.4}$$

- **El modelo**

La Figura 2.4 muestra el problema de programación lineal resultante.

$$\begin{array}{llllllllllll} Min & & & & & & & & & & & \\ 3x_{11} & +2x_{12} & +7x_{13} & +6x_{14} & +7x_{21} & +5x_{22} & +2x_{23} & +3x_{24} & +2x_{31} & +5x_{32} & +4x_{33} & +5x_{34} \\ x_{11} & +x_{12} & +x_{13} & +x_{14} & & & & & & & & \le 5,000 \\ & & & & x_{21} & +x_{22} & +x_{23} & +x_{24} & & & & \le 6,000 \\ & & & & & & & & x_{31} & +x_{32} & +x_{33} & +x_{34} \le 2,500 \\ x_{11} & & & & +x_{21} & & & & +x_{31} & & & = 6,000 \\ & x_{12} & & & & +x_{22} & & & & +x_{32} & & = 4,000 \\ & & x_{13} & & & & +x_{23} & & & & +x_{33} & = 2,000 \\ & & & x_{14} & & & & +x_{24} & & & & +x_{34} = 1,500 \\ & & & & & & & & & & & \\ x_{ij} \ge 0 & & \text{para} & i = 1,2,3 & \text{y} & \text{j=1,2,3,4} & & & & & & \end{array}$$

Figura 2.4 Representación del problema

2.1.2.2 Solución automatizada del ejemplo clásico del problema de transporte

Se utiliza el software LINDO (Linear Integer Discrete Optimizer) para resolverlo. En la Figura 2.5 se muestra la representación del problema como la relación entre cantidad a enviar, demanda y capacidad. En la Figura 2.6 se muestra el script proporcionado a LINDO y en la Figura 2.7 obtenemos la información de salida.

Datos de Entrada:

	1	2	3	4	Capacidad
1	?	?	?	?	5,000
2	?	?	?	?	6,000
3	?	?	?	?	2,500
Demanda	6,000	4,000	2,000	1,500	

Figura 2.5 Representación del problema

```
min 3x11+2x12+7x13+6x14+7x21+5x22+2x23+3x24+2x31+5x32+4x33+5x34
st
x11+x12+x13+x14<5000
x21+x22+x23+x24<6000
x31+x32+x33+x34<2500
x11+x21+x31=6000
x12+x22+x32=4000
x13+x23+x33=2000
x14+x24+x34=1500
end
```

Figura 2.6 Script proporcionado a LINDO.

Información de salida:

```
LP OPTIMUM FOUND AT STEP 6
OBJECTIVE FUNCTION VALUE                39500.00

VARIABLE        VALUE          REDUCED COST
X11      3500.000000          0.000000
X12      1500.000000          0.000000
X13         0.000000          8.000000
X14         0.000000          6.000000
X21         0.000000          1.000000
X22      2500.000000          0.000000
X23      2000.000000          0.000000
X24      1500.000000          0.000000
X31      2500.000000          0.000000
X32         0.000000          4.000000
X33         0.000000          6.000000
X34         0.000000          6.000000
```

Figura 2.7 Salida de LINDO

En la Figura 2.7, se aprecia que el costo mínimo de transporte es de 39,500 y las unidades a enviar por cada ruta, para que el costo de transporte sea el mínimo satisfaciendo las restricciones, son mostradas en la Figura 2.8:

	1	**2**	**3**	**4**	**Capacidad**
1	3,500	1,500			5,000
2		2,500	2,000	1,500	6,000
3	2,500				2,500
Demanda	6,000	4,000	2,000	1,500	

Figura 2.8 Unidades que minimizan la función objetivo

2.1.2.3 Elementos asociados a la programación lineal

Existen diversos elementos asociados a la programación lineal, que permiten obtener mayor conocimiento de la naturaleza del problema analizado. Los elementos son:
Los costos reducidos, las variables de holgura, los precios sombra, el análisis de sensibilidad de los coeficientes y el análisis de sensibilidad de las restricciones.

- **Costos Reducidos (REDUCED COST)**

Las variables diferentes de cero de la solución óptima tienen un costo reducido de CERO. Por ejemplo $x_{11}, x_{12}, x_{22}, x_{23}, x_{24}, x_{31}$

Para el resto de las variables el costo reducido (Figura 2.9) representa la penalidad que implicaría pagar si se introduce la variable a la solución. Si el tipo de objetivo es minimizar, entonces se debe entender la penalidad como un incremento en el costo de la función objetivo. Si el tipo de objetivo es maximizar, entonces se debe entender la penalidad como una disminución en el costo de la función objetivo.

VARIABLE	VALUE	REDUCED COST
X11	3,500.000000	0.000000
X12	1,500.000000	0.000000
X22	2,500.000000	0.000000
X23	2,000.000000	0.000000
X24	1,500.000000	0.000000
X31	2,500.000000	0.000000

Figura 2.9 Resultados de costos reducidos del ejemplo resuelto

Por ejemplo, si consideramos la variable x_{14} de nuestro ejemplo:

VARIABLE	VALUE	REDUCED COST
X14	0.000000	6.000000

Figura 2.10 Costo reducido de la variable x_{14} del ejemplo resuelto

Costo de la función objetivo:	:	39,500
Objetivo de la función objetivo	:	Minimizar
Costo Reducido	:	6

Si se introduce la variable x_{14} a la solución, entonces por cada unidad que se incremente esta variable el costo de la función objetivo sufre un incremento equivalente al costo reducido de la variable. En este caso como el costo reducido de la variable es 6, si se hace $x_{14} = 1$, la función objetivo se incrementa de 39,500.00 a 39,506.00.

- **Variables de holgura (SLACK OR SURPLUS)**

ROW	SLACK OR SURPLUS	DUAL PRICES
2)	0.000000	3.000000
3)	0.000000	0.000000
4)	0.000000	4.000000
5)	0.000000	-6.000000
6)	0.000000	-5.000000
7)	0.000000	-2.000000
8)	0.000000	-3.000000

Figura 2.11 Relación de variables de holgura

Las variables de holgura (Figura 2.11) indican en que medida la solución óptima satisface cada restricción:

- Para restricciones <= se refieren como SLACK
- Para restricciones >= se refieren como SURPLUS
- Si la restricción es exactamente satisfecha para la igualdad, entonces el valor de SLACK/SURPLUS es CERO.

Por ejemplo para la restricción 3, referenciada como ROW 4 (Figura 2.12)

```
ROW    SLACK OR SURPLUS      DUAL PRICES
4)          0.000000           4.000000
```

Figura 2.12 Variables de holgura de la restricción 3

Restricción 3

$$x_{31} + x_{32} + x_{33} + x_{34} \leq 2,500 \quad \text{.....(3)}$$

Sustituyendo los valores obtenidos en la figura 2.9, obtenemos:

$$\begin{array}{lllll} x_{31} & +x_{32} & +x_{33} & +x_{34} \leq & 2,500 \quad \text{.....(3)} \\ 2,500 & +0 & +0 & +0 \leq & 2,500 \quad \text{.....(3)} \\ 2,500 & & & \leq & 2,500 \quad \text{.....(3)} \end{array} \tag{2.5}$$

Se observa que el valor del lado izquierdo de la restricción es igual al valor del lado derecho de la restricción, lo cual significa que las unidades del recurso disponible de la restricción 3, están agotadas.

- Precios sombra (DUAL PRICES)

```
ROW          DUAL PRICES
2)     3.000000
3)     0.000000
4)     4.000000
5)    -6.000000
6)    -5.000000
7)    -2.000000
8)    -3.000000
```

Figura 2.13 Relación de precios sombra

Los precios sombra (Figura 2.13), representan la penalidad que implicaría pagar por cada unidad adicional del recurso disponible especificado en las restricciones. El incremento o decremento del costo de la función objetivo depende del tipo de objetivo y del valor positivo o negativo de los precios sombra.

A continuación, en la Figura 2.14, se describe el comportamiento de los precios sombra:

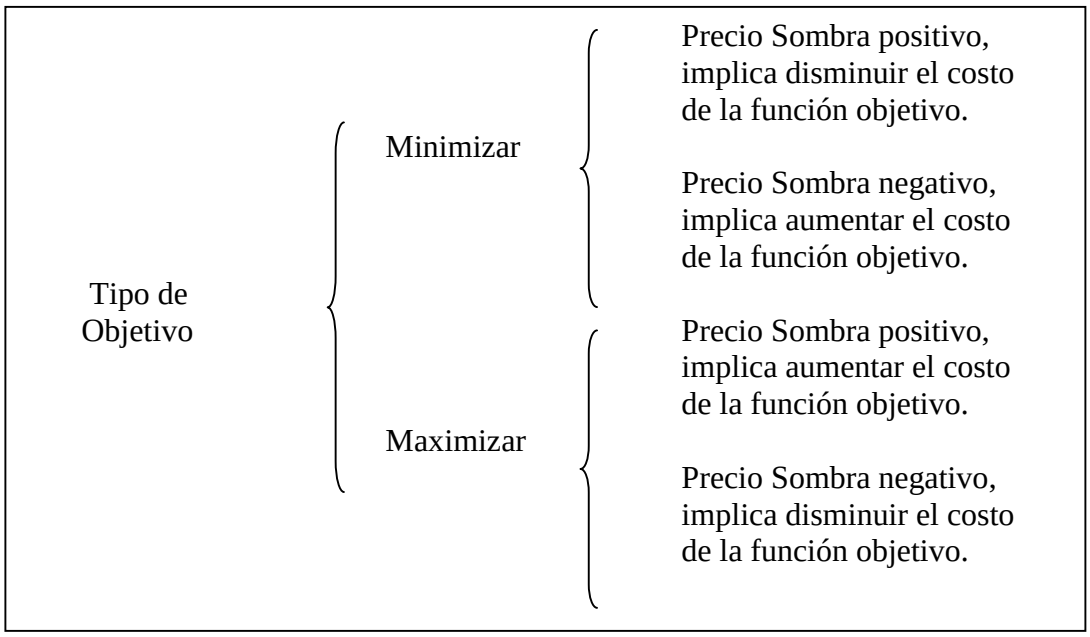

Figura 2.14 Comportamiento del uso de los precios sombra

Por ejemplo para la restricción 3 (Figura 2.15), referenciada como ROW 4

ROW	SLACK OR SURPLUS	DUAL PRICES
4)	0.000000	4.000000

Figura 2.15 Precio sombra de la restricción 3

Dado que el slack es CERO se concluye que ya no hay más recurso disponible, sin embargo si se incrementa en una unidad adicional dicho recurso (ya agotado), entonces el costo de la función objetivo disminuye en $4.

En tal caso la función objetivo disminuye de $39500.00 a $39496.00.

de

$$x_{31} + x_{32} + x_{33} + x_{34} \leq 2,500 \quad \text{.....(3)}$$

a

$$x_{31} + x_{32} + x_{33} + x_{34} \leq 2,501 \quad \text{.....(3)} \tag{2.6}$$

	Antes	Después
Costo de la función objetivo:	39,500	39,496

- **Análisis de la sensibilidad de los coeficientes de las variables de la función objetivo**

RANGES IN WHICH THE BASIS IS UNCHANGED:

	OBJ COEFFICIENT RANGES		
VARIABLE	CURRENT COEF	ALLOWABLE INCREASE	ALLOWABLE DECREASE
X11	3.000000	1.000000	4.000000
X12	2.000000	3.000000	1.000000
X13	7.000000	INFINITY	8.000000
X14	6.000000	INFINITY	6.000000
X21	7.000000	INFINITY	1.000000
X22	5.000000	1.000000	3.000000
X23	2.000000	6.000000	INFINITY
X24	3.000000	6.000000	INFINITY
X31	2.000000	4.000000	INFINITY
X32	5.000000	INFINITY	4.000000
X33	4.000000	INFINITY	6.000000
X34	5.000000	INFINITY	6.000000

Figura 2.16 Relación de análisis de sensibilidad de los coeficientes

El análisis de la sensibilidad de los coeficientes (Figura 2.16), permite saber las unidades que se pueden incrementar o decrementar los coeficientes de las variables de la función objetivo, sin causar un cambio en los valores de las variables en la solución óptima. Si consideramos la variable x_{12}

$$Min \quad \begin{aligned}&(3x_{11} + 2x_{12} + 7x_{13} + 6x_{14}) + (7x_{21} + 5x_{22} + 2x_{23} + 3x_{24}) \\ &+(2x_{31} + 5x_{32} + 4x_{33} + 5x_{34})\end{aligned} \tag{2.7}$$

```
RANGES IN WHICH THE BASIS IS UNCHANGED:

                          OBJ COEFFICIENT RANGES
 VARIABLE         CURRENT        ALLOWABLE        ALLOWABLE
                   COEF          INCREASE         DECREASE
      X12        2.000000         3.000000         1.000000
```

Figura 2.17 Análisis de sensibilidad del coeficiente de la variable x_{12}

El valor actual del coeficiente es de 2 (Figura 2.17). Entonces, este coeficiente puede ser incrementado un máximo de 3 y decrementado un máximo de 1, sin que cambien los valores de la solución óptima.

Por lo tanto, el rango de modificación del coeficiente de la variable x_{12} sin afectar los valores de las variables de la solución óptima es: $1 \le C_{12} \le 5$

Comprobación:

Resolviendo el mismo problema de la figura 2.4, pero incrementando en 1 el coeficiente de la variable x_{12} en la función objetivo:

$$Min \quad \begin{array}{l}(3x_{11}+3x_{12}+7x_{13}+6x_{14})+(7x_{21}+5x_{22}+2x_{23}+3x_{24}) \\ +(2x_{31}+5x_{32}+4x_{33}+5x_{34})\end{array} \tag{2.8}$$

En la Figura 2.18, se muestran los valores de salida obtenidos en la comprobación del análisis de sensibilidad de coeficientes, variable x_{12}:

```
LP OPTIMUM FOUND AT STEP      0
       OBJECTIVE FUNCTION VALUE
       1)      41000.00

 VARIABLE        VALUE          REDUCED COST
      X11      3500.000000          0.000000
      X12      1500.000000          0.000000
      X13         0.000000          7.000000
      X14         0.000000          5.000000
      X21         0.000000          2.000000
      X22      2500.000000          0.000000
      X23      2000.000000          0.000000
      X24      1500.000000          0.000000
      X31      2500.000000          0.000000
      X32         0.000000          3.000000
      X33         0.000000          5.000000
      X34         0.000000          5.000000
```

Figura 2.18 Valores de Salida

Comparando las figuras 2.7 y 2.18, podemos comprobar que los valores (columna VALUE) de las variables (columna VARIABLE) no cambiaron.

- **Análisis de la sensibilidad de los recursos disponibles (RHS)**

	RIGHTHAND SIDE RANGES		
ROW	CURRENT RHS	ALLOWABLE INCREASE	ALLOWABLE DECREASE
2	5000.000000	2500.000000	0.000000
3	6000.000000	INFINITY	0.000000
4	2500.000000	2500.000000	0.000000
5	6000.000000	0.000000	2500.000000
6	4000.000000	0.000000	2500.000000
7	2000.000000	0.000000	2000.000000
8	1500.000000	0.000000	1500.000000

Figura 2.19 Relación de análisis de sensibilidad de los recursos disponibles

El análisis de la sensibilidad de los recursos (Figura 2.19), permite determinar el número de unidades en que se pueden incrementar o decrementar el recurso de cada restricción (RHS), sin causar un cambio en los valores de las variables de la solución óptima.

Por ejemplo, si consideramos el renglón 3 (Figura 2.20), restricción 2, Capacidad de la planta 2:

	RIGHTHAND SIDE RANGES		
ROW	CURRENT RHS	ALLOWABLE INCREASE	ALLOWABLE DECREASE
3	6000.000000	INFINITY	0.000000

Figura 2.20 Análisis de sensibilidad de la restricción 2

El valor del Current RHS es de 6000. Entonces, el lado derecho de la restricción (Current RHS), puede ser incrementado un máximo de infinito y decrementado un máximo de 0.

Por lo tanto, el rango de modificación del lado derecho de la restricción sin afectar los valores de las variables de la solución óptima es:

$$6000 < \text{Capacidad de la Planta 2} < \text{Infinito}$$

Comprobación:

Usando el mismo problema de la Figura 2.4, pero con la restricción 2 alterada de la siguiente manera:

$$\begin{aligned} &\text{de} \\ &\quad x_{21} + x_{22} + x_{23} + x_{24} \leq 6,000 \quad \text{.....(2)} \\ &\text{a} \\ &\quad x_{21} + x_{22} + x_{23} + x_{24} \leq 7,000 \quad \text{.....(2)} \end{aligned} \tag{2.9}$$

Se obtienen los siguientes valores de salida:

```
LP OPTIMUM FOUND AT STEP      0

       OBJECTIVE FUNCTION VALUE

       1)      41000.00

 VARIABLE        VALUE          REDUCED COST
      X11      3500.000000          0.000000
      X12      1500.000000          0.000000
      X13         0.000000          7.000000
      X14         0.000000          5.000000
      X21         0.000000          2.000000
      X22      2500.000000          0.000000
      X23      2000.000000          0.000000
      X24      1500.000000          0.000000
      X31      2500.000000          0.000000
      X32         0.000000          3.000000
      X33         0.000000          5.000000
      X34         0.000000          5.000000
```

Figura 2.21 Valores de Salida

Comparando las figuras 2.7 y 2.21, podemos comprobar que los valores (columna VALUE) de las variables (columna VARIABLE) no cambiaron.

2.2 Matrices Dispersas

Una matriz dispersa, es aquella que contiene solo un pequeño número de elementos diferentes de cero. La Figura 2.22, muestra un ejemplo de una matriz dispersa [LINDO].

$$\begin{bmatrix} 3 & 0 & 0 & 2 \\ 0 & 26 & 0 & 39 \\ 4 & 5 & 8 & 0 \\ 0 & 7 & 1 & 0 \end{bmatrix}$$

Figura 2.22 Ejemplo de matriz dispersa

2.2.1 Representación de matrices dispersas mediante 3 vectores.

Debido a que al almacenar la matriz en memoria se desperdicia espacio con los elementos nulos, ésta se representa usando tres (u opcionalmente cuatro) vectores, de tal forma que los elementos nulos no se almacenan. Dado que la mayoría de los coeficientes de la matriz en modelos de programación matemática del mundo real son cero o nulos, este esquema de almacenamiento es muy adecuado para la representación en memoria de dichos modelos [LINDO].

2.2.2 Procedimiento de representación de matrices dispersas mediante 3 vectores

Primero se crea un vector denominado vector de Valores, el cual contiene todos los elementos distintos de cero de la matriz, ordenados por la columna. Para la matriz, del ejemplo anterior, este vector sería: [3 4 26 5 7 8 1 2 39].

Los elementos 3 y 4 de la primera columna aparecen primero, cuyos índices son el 0 y el 1 respectivamente. Después aparecen los elementos 26, 5 y 7 de la segunda columna, cuyos índices son el 2, 3 y 4 respectivamente. Aparecen después los elementos 8 y 1 de la tercera columna, cuyos índices son el 5 y 6 respectivamente. Finalmente aparecen los elementos de la cuarta columna 2 y 39, cuyos índices son el 7 y 8 respectivamente. Como se observa en la construcción del vector de Valores todos los ceros de la matriz dispersa son descartados.

Ahora se crea un vector denominado Inicio-Columna, en el cual se registra qué índices del vector de Valores representan el comienzo de una columna de la matriz original.

Para el vector de Valores anterior el vector Inicio–Columna es: [0 2 5 7 9].

Este vector contiene un elemento adicional, el cual indica en que índice del vector de Valores termina la última columna. Éste último elemento es igual a la longitud del vector de Valores. El vector Inicio-Columna permite determinar a que columna pertenece cada elemento del vector de Valores.

Una vez realizado el proceso anterior, la única información adicional que se requiere para especificar completamente la matriz dispersa, es el número de fila de cada elemento del vector de Valores. Esta información se almacena en un tercer vector, el vector Índice-Fila. Este vector es de la misma longitud que el vector de Valores y cada uno de sus elementos indica la fila a la que pertenece el elemento correspondiente del vector Valores. Por ejemplo, el primer elemento del vector de Valores, el número 3, pertenece a la fila 0, así que el primer elemento del vector Índice-Fila es 0. De igual forma, el segundo elemento del vector de Valores es 4 y pertenece a la fila 2, así que el segundo elemento del vector Índice-Fila es 2. De manera similar se determinan todos los elementos del vector Índice - Fila [0 2 1 2 3 2 3 0 1].

2.3 Interfase de programación de LINDO 2.0

La interfase de programación de LINDO 2.0 es un conjunto de funciones que pueden ser invocadas desde diversos lenguajes de programación para resolver problemas de programación lineal. Para demostrar como se resuelve un problema de programación lineal usando esta interfase, se utilizará el ejemplo de la Figura 2.23.

$$
\begin{array}{llrrrcr}
max & z = & 10x_{11} & +150x_{12} & +30x_{13} & & \\
st & & & & & & \\
 & & x_{11} & +3x_{12} & & \leq & 100 \\
 & & 10x_{11} & +10x_{12} & +5x_{13} & \geq & 1000 \\
 & & x_{11} & & & \leq & 200 \\
 & & & x_{12} & & \leq & 20 \\
 & & & & x_{13} & \leq & 50
\end{array}
$$

Figura 2.23 Ejemplo de un problema de programación lineal

2.3.1 Representación en la matriz de restricciones

El primer paso para resolver un problema de programación lineal es representar la matriz de restricciones utilizando la representación de tres vectores. En la Figura 2.24, indica la matriz de restricciones del problema considerado:

$$\begin{bmatrix} 1 & 3 & 0 \\ 10 & 10 & 5 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Figura 2.24 Matriz de restricciones del problema considerado

Entonces su representación mediante tres vectores es la siguiente:

vector de Valores: [1 10 1 3 10 1 5 1]

vector Inicio-Columna: [0 3 6 8]

vector Índice -Fila: [0 1 2 0 1 3 1 4]

2.3.2 Estructuras de datos utilizadas por el API de LINDO

Una vez determinada la representación mediante tres vectores de la matriz de restricciones, todos los componentes del problema se definen a través de las estructuras de datos que se describen en la Tabla 2.1.

La primera columna de la tabla, contiene el nombre de la estructura requerida por LINDO; la segunda, el tipo de dato; la tercera, la descripción del componente del problema que se registra en la estructura y la última, muestra el contenido correspondiente a los componentes del ejemplo considerado.

Tabla 2.1 Estructuras de datos utilizadas por la interfaz de LINDO

Dato	Tipo	Descripción	Ejemplo
pachContypes	Char *	Tipo de restricción	L, G, L, L
nCons	Int	Número de restricciones	4
padC	double*	Coeficientes de la función objetivo	10, 150,30
nAnnz	Int	Número de no ceros en la matriz de restricciones	8
padAcoef	double*	Coeficientes de los no ceros en la matriz de restricción	1, 10, 1, 3, 10, 1, 5, 1
paiArows	Int *	Índice de fila de los no ceros	0, 1, 2, 0, 1, 3, 1, 4
nVars	Int	Número de variables	3
paiAcols	Int *	Índice del primer no ceros en cada columna	0, 3, 6
padB	double*	Coeficientes del lado derecho de las restricciones	100, 1000, 200, 20, 50
padL	double*	Límites bajos de cada variable	Null
padU	double*	Límites altos de cada variable	Null
dObjconst	double	Valor constante agregado a la función objetivo	0
pacAcols	Int *	Longitud de cada columna	null
tipo_var	Int *	Tipo de variable, 1 si es entera, 0 si no lo es	1

2.3.3 Codificación en lenguaje C.

Para resolver el problema de programación lineal, utilizando el lenguaje de programación C, se deben realizar los siguientes pasos:

1. Crear un entorno de LINDO.
2. Crear un modelo en el entorno.
3. Definir el modelo usando las estructuras descritas.
4. Ejecutar la optimización.
5. Recuperar los resultados de la optimización.
6. Eliminar el entorno de LINDO.

La Figura 2.25 muestra el código fuente correspondiente al ejemplo considerado. Mayores detalles relativos a este proceso se pueden encontrar en [LINDO].

```
#include <stdlib.h>   #include <stdio.h>
#include "lindo.h"    #include "license.h"
#define APIERRORSETUP  \
   int nErrorCode; \
   char cErrorMessage[LS_MAX_ERROR_MESSAGE_LENGTH] \
#define APIERRORCHECK  \
   if (nErrorCode) \
   { \
      if ( pEnv) \
      { \
         LSgetErrorMessage( pEnv, nErrorCode, \
          cErrorMessage); \
         printf("Código de error=%d:  %s\n", nErrorCode, \
          cErrorMessage); \
      } else {\
         printf( "Error fatal\n"); \
      } \
      exit(1); \
   } \
int main() {
   APIERRORSETUP;
/* Número de restricciones */      int nM = 4;
/* Número de variables */       int nN = 3;
/* tipo entorno de LINDO */      pLSenv pEnv;
/* tipo modelo de LINDO*/   pLSmodel pModel; int nSolStatus;
```

Figura 2.25 Código fuente del problema considerado

```
/* 1. Crear un entorno de LINDO */

   pEnv = LScreateEnv ( &nErrorCode, MY_LICENSE_KEY);
   if ( nErrorCode == LSERR_NO_VALID_LICENSE)
   {
      printf( "Licencia invalida!\n");
      exit( 1);
   }
   APIERRORCHECK;

/* 2. Crear un modelo en el entorno*/

   pModel = LScreateModel ( pEnv, &nErrorCode);
   APIERRORCHECK;
   {

/* 3. Definir el modelo */

/* Dirección de optimización*/
      int nDir = LS_MAX;
/* Término constante en la función objetivo*/
      double dObjConst = 0.;
/* Coeficientes de la función objetivo */
      double adC[2] = { 10., 150., 30.};
/* Lados derechos de las restricciones*/
      double adB[3] = { 100., 1000., 200., 20.};
/* Tipos de restricciones*/
      char acConTypes[3] = {'L', 'G', 'L', 'L'};
/* Número de elementos no cero en la matriz de
restricciones*/
      int nNZ = 8;
/* Vector Valores de la matriz de restricciones */
      double adA[4] = { 1., 10., 1., 3., 10., 1., 5., 1.};
/* Vector Inicio-Columna de la matriz de restricciones*/
      int anBegCol[3] = { 0, 3, 6, nNZ};
/* Vector Índice-Fila de la matriz de restricciones*/
      int anRowX[4] = { 0, 1, 2, 0, 1, 3, 1, 4};
/* Longitud de cada columna (se debe especificar cuando la
representación de la matriz de restricciones incluya
elementos que son cero, en otro caso se le asigna NULL)*/
      int *pnLenCol = NULL;
```

Figura 2.25 Continuación …

```
/* Limites inferiores y superiores de las variables de
decisión del   problema. Los valores por omisión son cero e
infinito. */
double *pdLower = NULL, *pdUpper = NULL;
/* Cargar el modelo a memoria utilizando la función
LSloadLPData */
      nErrorCode = LSloadLPData( pModel, nM, nN, nDir,
       dObjConst, adC, adB, acConTypes, nNZ, anBegCol,
       pnLenCol, adA, anRowX, pdLower, pdUpper);
      APIERRORCHECK;
   }
 /* 4. Ejecutar la optimización */
   nErrorCode = LSoptimize( pModel,
    LS_METHOD_PSIMPLEX, &nSolStatus);
   APIERRORCHECK;
   if (nSolStatus == LS_STATUS_OPTIMAL ||
       nSolStatus == LS_STATUS_BASIC_OPTIMAL)
   {
 /* 5. Recuperar los resultados de la optimización */
      int i;
      double adX[ 2], dObj;
 /* Obtener el valor óptimo de la función objetivo*/
      nErrorCode = LSgetInfo( pModel, LS_DINFO_POBJ, &dObj)
;
      APIERRORCHECK;
      printf( "Valor objetivo óptimo= %g\n", dObj);
 /* Obtener los valores de las variables de decisión del
problema*/
      nErrorCode = LSgetPrimalSolution ( pModel, adX);
      APIERRORCHECK;
      printf ("Primal values \n");
      for (i = 0; i < nN; i++)
printf( " x[%d] = %g\n", i,adX[i]); printf ("\n");
   }
   else   {
     printf( "Solución óptima no alcanzada -- código: %d\n",
     nSolStatus);
   }
 /* 6. Eliminar el entorno de LINDO */
   nErrorCode = LSdeleteEnv( &pEnv);
   printf("Presione <Enter> ...");
   getchar();
}
```

Figura 2.25 Continuación …

2.4 Trabajos relacionados

Dado que el problema de abastecimiento internacional esta muy relacionado con el problema de ubicación de plantas, a continuación se describen algunos de los trabajos en los que se aborda el problema de la ubicación de plantas:

Kraup y Pruzan, hicieron una investigación del problema de ubicación de plantas, considerando las siguientes versiones: no capacitada, capacitada, dinámica (multiperiodo) y estocástica. Una versión estocástica, es mencionada muy ligeramente y no reportan resultados [Kraup 1979].

Verter y Dincer, analizan el problema ubicación de plantas y consideran las siguientes versiones del problema: no capacitada, capacitada, estocástica y el problema de la ubicación internacional de las plantas [Verter 1992].

Los siguientes trabajos abordan la versión estocástica del problema de la ubicación internacional de las plantas.

Louveaux y Peters, presentaron un problema basado en escenarios en el cual la capacidad de las plantas se determina como resultado de una decisión inicial. A pesar de la limitación de la capacidad de las plantas, los autores lo refieren como un problema de capacidad infinita [Louveaux 1992].

Jucker y Carlson, analizan el problema considerando un solo producto, un solo periodo y que el precio y la demanda son inciertas. Presentan dos tipos diferentes de plantas: las que son capaces de controlar el precio y la demanda de sus productos y las que no tienen control de estos factores. La solución del problema de localización de las plantas requiere la solución de un modelo de programación cuadrática entera. Para simplificar la solución de este problema se introduce el concepto de planta dominante, donde se considera que cada uno de los mercados solo puede ser atendido por una sola planta. Este supuesto permite transformar la parte cuadrática del problema en un subproblema lineal [Jucker 1976].

Hodder y Jucker, presentan un modelo de un único periodo, un sólo producto. Modelan un control de precios y una demanda que depende del precio y resuelven el problema utilizando plantas dominantes [Hodder 1982].

Hodder y Dincer, conciben simultáneamente el problema de la ubicación de las plantas y las decisiones de financiamiento. La rentabilidad de una planta depende de la ubicación de las plantas. Formulan un modelo de un solo producto y un solo periodo. Suponen que toda la producción de las plantas, hasta un límite determinado, se podrá vender con una ganancia unitaria incierta [Hodder 1986].

Gutiérrez y Kouvelis, refieren el problema de la generación de escenarios para modelar precios inciertos y resolver el problema simple de ubicación de plantas no capacitado. Esto es equivalente a un modelo del problema del abastecimiento internacional con un criterio regret mínimax. Sin embargo, sus resultados deben ser examinados detalladamente, ya que las soluciones reportadas pueden ser peores que el valor esperado de las soluciones correspondientes a cada uno de los escenarios, cuando se utiliza un conjunto grande de escenarios [Gutiérrez 1995].

Lawrence y Buss, consideran la posibilidad de producir en diferentes países para obtener ventaja de las variaciones de la tasa de cambio de las monedas locales. Modelan la producción de un único producto en dos países diferentes, cada uno de los cuales tiene una demanda determinada [Lawrence 1999].

La Tabla 2.2, muestra un resumen de las principales características de los trabajos anteriores.

Tabla 2.2 Resumen de artículos que tratan el problema de la ubicación de la planta.

Autores	Robusto	Periodo	Producto	Capacidad finita	Estocástico
Louveraux y Peters, 1992	escenarios	único	único	sí	sí
Jucker y Carlson, 1976		sí	sí	sí	sí
Hodder y Jucker, 1982,1985a, 1985b		sí	sí		
Haug, 1985		Múltiple	Único		
Hodder y Dincer, 1986		Único	Único		
Cohen y Lee, 1989		Único	Múltiple		
Gutierrez y Kouvelis, 1995				Infinita	

Algunos de los trabajos en que se aborda el problema del abastecimiento internacional con incertidumbre son los siguientes:

Kouvelis y Yu, tratan sobre el problema de abastecimiento internacional con capacidad infinita en los proveedores. Aplican un enfoque robusto en la formulación del problema y utilizan una función objetivo de tipo minimax. El método de

solución que proponen consiste en encontrar la solución para el peor de los escenarios posibles. La principal desventaja de este enfoque es que la solución que se produce es demasiado pesimista y en general se pudiera encontrar una mejor solución en base a un escenario promedio [Kouvelis y Yu 1997].

Escudero, presentó un trabajo relacionado al abastecimiento en los procesos de manufactura. Donde el costo esperado es el único término a optimizar en la función objetivo [Escudero 1993].

Del problema robusto del abastecimiento internacional con capacidad finita, se conocen los siguientes trabajos:

La formulación robusta del problema ROCIS fue propuesta por primera vez por González- Velarde y Laguna en [González 2004]. González Velarde y Laguna proponen un método de solución que incorpora mecanismos de búsqueda tabú. El proceso consiste en construir una solución inicial seguida de una búsqueda voraz en la vecindad de dicha solución. Como la elección de la solución inicial determina la eficiencia del proceso, dicha solución se construye dando preferencia a los proveedores de mayor capacidad y con un menor costo fijo.

Por otra parte, González-Velarde y Martí en [González 2006], proponen un método de solución basado en reencadenamiento de trayectorias. El método, incorpora el costo de envío de cada proveedor a todas las plantas, como parte de la estrategia de prioridades que se utiliza para construir un conjunto de soluciones iniciales. Los propios autores señalan, que esta manera de incorporar el costo de envío parece ser demasiado pesimista. Los resultados presentados en el artículo, no permiten determinar en que medida la mejora reportada depende del cambio en la estrategia de prioridades para la generación de soluciones iniciales.

La Tabla 2.3, muestra las estrategias de selección de proveedores, utilizadas en los métodos de solución del problema de abastecimiento internacional con capacidad finita reportados en la literatura.

Tabla 2.3 Características relevantes

Artículo	**Método de solución**	**Estrategia**
Laguna y Velarde, 2004	Búsqueda Tabú.	$G_i = \frac{f_i}{b_i}$
Marti y Velarde, 2006	Re-encadenamiento de trayectorias	$G_i = \frac{f_i + \left(\sum_{s \in S} p_s \left(\sum_{j=1}^{n} c_{ij} e_{is} \right) \right)}{b_i}, \forall j \in N, i \in M$

La estrategia de selección de proveedores permite la construcción de soluciones iniciales de calidad, lo cual determina el desempeño del método de solución. Una de las cuestiones actualmente abiertas en el estudio de este problema es la determinación de nuevas estrategias de prioridades que permitan mejorar el desempeño de dicha solución [González 2006].

En esta tesis, se proponen diferentes estrategias de prioridades para la construcción de soluciones iniciales y se presenta evidencia experimental del impacto en el desempeño de la solución tabú del problema ROCIS.

Capítulo 3

SOLUCIÓN DE REFERENCIA

En esta sección se describe detalladamente la implementación en lenguaje C de la solución tabú reportada en [González 2004]. El desempeño de esta solución será utilizado como referencia para evaluar el desempeño de las soluciones propuestas.

CONTENIDO

CONTENIDO *36*
FIGURAS *36*
TABLAS *36*
3.1 Descripción general *33*
Figura 3.1 Continuación … *36*
3.2 Representación binaria y decimal de las soluciones *36*
3.3 Estructuras de datos utilizadas *37*
3.4 Funciones utilizadas *43*
3.4.1 GeneraciónVoraz() 43
3.4.1.1 Función Initialize() 44
3.4.1.2 Función select_min() 45
3.4.2 Función BúsquedaLocal() 46
3.4.2.1 Función form_list() 48
3.4.2.2 Función select_move() 55
3.4.2.3 Algoritmo execute_move 61

FIGURAS

Figura 3.1 Continuación …. 34
Figura 3.1 Continuación…. 35
Figura 3.2 Algoritmo de la función Initialize() 44
Figura 3.2 Continuación …. 45
Figura 3.3 Algoritmo de la función select_min() 46
Figura 3.4 Algoritmo de la función BusquedaLocal () 48
Figura 3.5 Algoritmo de la función form_list () 52
Figura 3.5 Continuación …. 53
Figura 3.5 Continuación …. 54
Figura 3.5 Continuación …. 55
Figura 3.6 Algoritmo *select_move* 58
Figura 3.6 Continuación …. 59
Figura 3.6 Algoritmo *select_move* 60
Figura 3.7 Algoritmo execute_move 61
Figura 3.7 Continuación …. 62

TABLAS

Tabla 3.1 Estructuras utilizadas por la interfaz de LINDO 37
Tabla 3.2 Estructura sInterface: datos de entrada para el proceso de optimización 38
Tabla 3.3 Estructura sInterface: información de salida del proceso de optimización que se genera al invocar la función de LINDO GetInfo(). 38
Tabla 3.3 Continuación … 39
Tabla 3.4 Estructura sInterfaceComunes: datos de entrada al proceso de optimización que no son dependientes de la solución analizada o del escenario. 39
Tabla 3.5 Estructura inputdata 40

Tabla 3.6 Estructura soldata: información de la solución o selección de proveedores analizada que se genera en el proceso de solución 40
Tabla 3.6 Continuación … 41
Tabla 3.7 Estructura movedata: información relativa al movimiento (inserción, eliminación o intercambio de proveedores) realizado para construir una solución actual. 41
Tabla 3.7 Continuación … 42
Tabla 3.8 Funciones utilizadas para resolver el problema ROCIS. 43
Tabla 3.9. Funciones utilizadas por la función GeneraciónVoraz(). 44
Tabla 3.10 Listas globales que permiten mantener un registro histórico de las soluciones evaluadas en el proceso. 47
Tabla 3.11 Funciones utilizadas en la función BúsquedaLocal() 47
Tabla 3.12 Listas de movimientos de inserción, eliminación e intercambio. 49
Tabla 3.13 Listas tabú de inserción, eliminación e intercambio. 49
Tabla 3.13 Continuación. 50

3.1 Descripción general

El método reportado en [González 2004] consiste en generar una solución inicial, construir una vecindad de soluciones y realizar una búsqueda exhaustiva en la vecindad. La primera solución inicial, se construye dando prioridad a los proveedores de menor costo fijo y mayor capacidad de producción. Para toda solución inicial se resuelven los subproblemas lineales de transporte asociados a cada escenario y se determina el valor de la función objetivo correspondiente. Los valores esperados de los precios sombra de las soluciones de cada uno de los subproblemas lineales, son utilizados para determinar un nuevo conjunto de posibilidades de selección de proveedores. Estas elecciones potenciales se combinan para formar una vecindad de soluciones prometedoras y se lleva a cabo una búsqueda exhaustiva en la vecindad generada. El método de búsqueda, utiliza la estrategia de memoria de corto plazo de tipo búsqueda tabú [Glover 1997]. A medida que la búsqueda avanza se actualiza la mejor solución encontrada y se continúa hasta terminar de revisar la vecindad. Una vez que finaliza la búsqueda se actualiza la solución inicial, con la mejor solución encontrada y se continúa el proceso durante 50 iteraciones. La Figura 3.1 muestra el pseudocódigo del algoritmo de la solución de referencia.

Línea	Descripción
1	Cargar datos de la instancia
2	Generar la primera solución inicial
3	Calcular la demanda máxima $D = \max_{s \in S}(\sum_{j \in N} d_{js})$
4	Ordenar los proveedores de manera ascendente por $G_i = \frac{f_i}{b_i}$
5	Seleccionar los proveedores iniciando con el primero de la lista ordenada hasta que la suma de las capacidades de los proveedores seleccionados sea mayor que D.

Figura 3.1 Algoritmo de la solución referencia

Línea	Descripción
6	Determinar F[y], resolviendo los subproblemas de distribución que se generan en todos los escenarios.
7	Hacer 50 iteraciones
8	Determinar el costo relativo de los proveedores (r_i)
9	Determinar el valor esperado de los precios sombra ($®_{is}$) asociados a la restricción correspondiente a cada proveedor, en la solución de los subproblemas asociados a la solución inicial (y) en todos los escenarios posibles.
10	$E(®_i) = \sum_{s \in S} p_s ®_{is}$
11	Calcular el valor r_i de cada proveedor.
12	$r_i = \begin{cases} \frac{E(®_i)}{f_i} & si \quad E(®_i) < 0 \\ f_i & si \quad E(®_i) = 0 \end{cases}$
13	Para todas las posibles inserciones, eliminaciones e intercambios de proveedores que se pueden realizar a partir de y, se valida que la configuración correspondiente y' sea factible con respecto a la demanda máxima.
14	Se integran las listas de movimientos de inserción, eliminación e intercambio, a partir de los movimientos candidatos identificados en el paso anterior si los proveedores involucrados no forman parte de la lista tabú correspondiente al tipo de movimiento.
15	La lista de movimientos de inserción contiene los 3 proveedores de menor valor r_i.

Figura 3.1 Continuación …

Línea	Descripción
16	La lista de movimientos de eliminación contiene los 3 proveedores de mayor valor r_i.
17	La lista de movimientos de intercambio contiene los $\frac{(m^2-m)}{8}$ proveedores con el menor valor $r_j - r_i$ correspondiente al intercambio del proveedor i por el proveedor j en la solución inicial (configuración y).
18	Para cada configuración y' generada a partir de los movimientos de las listas de movimientos de inserción, eliminación e intercambio:
19	Calcular el valor de la función objetivo $$F(y) = \sum_{s \in S} p_s \left(\sum_{i \in M} e_{is} f_i y_i + z_s \right) + w \sqrt{\frac{\sum_{s \in S^+} p_s \left(z_s - E(z_s) \right)^2}{\sum_{s \in S^+} p_s}}$$ donde $S^+ = \{s : z_s - E(z_s) \geq 0\}$
20	Los proveedores involucrados en el movimiento utilizado para generar la configuración y' se incorporan a la lista tabú de inserción (si el movimiento fue de eliminación), eliminación (si el movimiento fue de inserción) o intercambio. El número de iteraciones que se considera tabú a los proveedores involucrados, en el movimiento, es de $\frac{m}{3}$ para inserciones y eliminaciones; y de $\frac{m(m-1)}{16}$ para intercambios .
21	Actualizar la mejor solución encontrada (y_{mejor}).

Figura 3.1 Continuación…

Línea	Descripción
22	Actualizar la solución inicial (y) con la mejor solución encontrada.
23	Actualizar la mejor solución global
24	Regresar a la línea 8, hasta realizar las 50 iteraciones
25	Reportar la mejor solución global

Figura 3.1 Continuación …

3.2 Representación binaria y decimal de las soluciones

Por definición la representación de una solución y es el vector binario y_i, donde $i = 1, 2, ..., m$ en el que se registra una selección de proveedores determinada. Entonces, una forma alterna de presentarla es utilizar su representación decimal $H(y) = \sum_{i \in M} y_i' 2^i$. Por ejemplo, la representación decimal de la solución binaria $y = \{0, 0, 1\}$ es la siguiente:

$$H(y) = \sum_{i \in M} y_i' 2^i = y_1' 2^1 + y_2' 2^2 + y_3' 2^3 = \cancel{0x2} + \cancel{0x4} + 1x8 = 8$$

Este mecanismo de representación alterno es utilizado en la implementación de la solución tabú para mejorar su eficiencia. En primer término, se observa que uno de los procesos de mayor costo computacional es la evaluación de las soluciones, ya que cada evaluación requiere invocar la función de optimización de LINDO 27 veces. Para minimizar el número de invocaciones de este tipo, se genera un registro histórico de las soluciones evaluadas. Sí para llevar este registro se utiliza la representación binaria, el consumo de memoria sería demasiado grande.

Por lo tanto resulta más económico realizar este registro histórico utilizando la representación decimal $H(y)$ y solo utilizar la representación binaria para registrar las soluciones óptimas (locales o globales).

Con este mecanismo, antes de evaluar una nueva solución vecina, se verifica si no ha sido evaluada anteriormente utilizando el registro histórico de la

representación decimal de las soluciones evaluadas, si esto ocurre se realiza una nueva evaluación y en otro caso simplemente se recupera del registro histórico el valor de la solución.

3.3 Estructuras de datos utilizadas

Para la solución del problema ROCIS se utilizan las estructuras que se indican en la Tabla 3.1. La Tabla 3.2 muestra los componentes de la estructura sInterface usados para cargar los datos del modelo a evaluar. La Tabla 3.3 enseña los componentes de la estructura sInterface usados para recibir la información generada por LINDO después de evaluar el modelo. La Tabla 3.4 indica los componentes de la estructura sInterfaceComunes. La Tabla 3.5 señala los componentes de la estructura inputdata. La Tabla 3.6 presenta los componentes de la estructura soldata. La Tabla 3.7 muestra los componentes de la estructura movedata.

Tabla 3.1 Estructuras utilizadas por la interfaz de LINDO

Nombre	**Descripción**
sInterface	Estructura que se utiliza para almacenar los datos de entrada y la información de salida del proceso de optimización. En esta estructura se almacenan únicamente los datos de entrada que son dependientes de la solución (selección de proveedores) considerada o del escenario.
sInterfaceComunes	En esta estructura se almacenan únicamente los datos de entrada que no son dependientes de la solución (selección de proveedores) considerada o del escenario.
inputdata	Guarda todos los parámetros de la instancia analizada.
soldata	Guarda la información de la solución o selección de proveedores analizada.
movedata	Guarda la información del movimiento (inserción, eliminación o intercambio de proveedores) que genera un óptimo local.

Tabla 3.2 Estructura sInterface: datos de entrada para el proceso de optimización

Nombre	Descripción
double *padB;	Lista que contiene los lados derechos de las restricciones del modelo matemático a resolver. Su tamaño es la suma del número de plantas más el número de proveedores de la instancia analizada. La lista se divide en dos bloques, el primero tiene los valores de las demandas de las plantas en el escenario analizado y el segundo bloque tiene los valores de las capacidades de los proveedores seleccionados de la solución considerada.
double *padC;	Lista que contiene los coeficientes de la función objetivo del modelo. Su tamaño es el producto del número de plantas por el número de proveedores.

Tabla 3.3 Estructura sInterface: información de salida del proceso de optimización que se genera al invocar la función de LINDO GetInfo().

Nombre	Descripción
int nSolStatus;	Contiene el resultado del proceso de optimización del modelo matemático realizado por LINDO. Los valores LS_STATUS_OPTIMAL / LS_STATUS_BASIC_OPTIMAL indican que la solución del modelo es factible, cualquier otro valor significa que no existe una solución factible.
double *padX;	Lista que contiene los valores de la solución óptima (primal) del modelo (x_{ijs}). Su tamaño es el número de coeficientes de la función objetivo del modelo.
double *padSlacks;	Lista que contienen los valores de las variables de holgura (SLACKS/SURPLUS). Su tamaño es igual al número de restricciones del modelo.

Tabla 3.3 Continuación …

Nombre	Descripción
double *padDual;	Lista que contiene los valores de los precios sombra (duales). Su tamaño es igual al número de restricciones del modelo.
double dObj;	Contiene el valor óptimo de la función objetivo del modelo matemático evaluado.

Tabla 3.4 Estructura sInterfaceComunes: datos de entrada al proceso de optimización que no son dependientes de la solución analizada o del escenario.

Nombre	Descripción
int nM;	Número de proveedores de la instancia analizada
int nN;	Número de plantas de la instancia analizada
int nCoefficients;	Número de coeficientes de la función objetivo del modelo matemático a optimizar. Es igual al producto del número de proveedores por el numero de plantas
int nConstraints;	Número de restricciones del modelo, su valor es la suma del número de proveedores más el número de plantas
int nDir;	Valor que indica el tipo de objetivo: minimizar (LS_MIN) ó maximizar (LS_MAX).
int nNZ;	Número de elementos diferentes de ceros de la matriz de coeficientes de las restricciones del modelo.
int *panRowX;	Vector de Valores de la matriz de restricciones.
int *panBegCol;	Vector de Inicio Columna de la matriz de restricciones
double *padA;	Vector Índice-Fila de la matriz de restricciones
char *pacConTypes;	Vector que contiene la dirección de las relaciones de orden de las restricciones. Por ejemplo: G si la relación es de mayor o igual, y L si es de menor o igual.

Tabla 3.5 Estructura inputdata

Nombre	**Descripción**
int plants;	Número de plantas
int supp;	Número de proveedores
int scen;	Número de escenarios (27)
double **trancost;	Costos de transporte de producto desde la localidad del proveedor hacia la localidad de la planta

Tabla 3.6 Estructura soldata: información de la solución o selección de proveedores analizada que se genera en el proceso de solución

Nombre	**Descripción**
int *cap;	Capacidades de abastecimiento de los proveedores
double *fixedcost;	Costo fijo del producto abastecido por los proveedores
int **demand;	Demanda de producto de las plantas en cada escenario
double **exch;	Tasa de cambio de la moneda en la localidad de los proveedores para cada escenario
double *prob;	Probabilidades de ocurrencia de los escenarios
double max_dem;	Demanda máxima requerida por las plantas considerando todos los escenarios.
int *y;	Vector binario que contiene la selección de proveedores o solución actual. Si el proveedor i esta seleccionado la posición i del vector y tiene valor 1, en otro caso el valor de dicha posición es 0.
double exp_value;	Valor esperado, es el primer término de la función objetivo del problema. Asociado a la solución y evaluada: $\sum_{s \in S} p_s \left(\sum_{i \in M} e_{is} f_i y_i + z_s \right)$

Tabla 3.6 Continuación …

Nombre	**Descripción**
double bstdv;	Desviación estándar, es el segundo término de la función objetivo del problema. Asociado a la solución y evaluada: $\sqrt{\frac{\sum_{s\in S^+} p_s\left(z_s - E\left(z_s\right)\right)^2}{\sum_{s\in S^+} p_s}}$

Tabla 3.7 Estructura movedata: información relativa al movimiento (inserción, eliminación o intercambio de proveedores) realizado para construir una solución actual.

Nombre	**Descripción**
double av_trans;	Elemento del primer término de la función objetivo del problema. Asociado a la solución y evaluada: $\sum_{s\in S} p_s\left(z_s\right)$
double av_fixed;	Elemento del primer término de la función objetivo del problema. Asociado a la solución y evaluada: $\sum_{s\in S} p_s\left(\sum_{i\in M} e_{is} f_i y_i\right)$
int sol_cap;	Capacidad de la solución actual.
double **dual_cost;	Matriz que contiene los precios sombra de la solución y evaluada.
double *exp_dual_cost;	Vector que contiene los valores esperados de los precios sombra asociados a cada proveedor en todos los escenarios, para la solución y evaluada. $E(®_i) = \sum_{s\in S} p_s ®_{is}$
int *ybest;	Vector binario que contiene la solución óptima global.
double best_value;	Valor de la función objetivo correspondiente a la solución óptima global $F(y_{best})$.

Tabla 3.7 Continuación …

Nombre	**Descripción**
double **best_shadow;	Matriz que contiene los precios sombra de la solución y_{best} .
int best_iter;	Número de iteraciones requeridas para encontrar la mejor solución.
clock_t itime;	Tiempo inicial del proceso de solución de la instancia.
clock_t btime;	Tiempo requerido para encontrar la mejor solución de la instancia.
int type;	Tipo de movimiento realizado para generar solución actual. 1 para inserción, 2 para eliminación y 3 para intercambio
int flag;	Indica si la solución actual ya había sido evaluada o no.
int insertion;	Número del proveedor insertado en la solución actual.
int deletion;	Número del proveedor eliminado de la solución actual
double value;	Valor de la función objetivo del modelo matemático asociado a la solución actual.
int *iter_in;	Lista que contiene los números de los proveedores que no se pueden insertar en la solución actual (Lista Tabú de movimientos de inserción).
int *iter_out;	Lista que contiene los números de los proveedores que no se pueden eliminar de la solución actual (Lista Tabú de movimientos de eliminación).
Int **iter_swap;	Lista que contiene los pares de proveedores que no se pueden intercambiar en la solución actual (Lista Tabú de movimientos de intercambio).

3.4 Funciones utilizadas

La Tabla 3.8 contiene la descripción de las funciones más importantes que se utilizan en la implementación de la solución tabú del problema ROCIS.

Tabla 3.8 Funciones utilizadas para resolver el problema ROCIS.

Función	Descripción
GeneraciónVoraz()	Genera la primera solución actual seleccionando primero los proveedores de menor costo fijo y mayor capacidad de producción. Esta función corresponde a las líneas 2, 3, 4 y 5 de la Figura 3.1
BúsquedaLocal()	A partir de la solución actual construye una vecindad de soluciones con base en el valor esperado de los precios sombra asociados a la solución inicial. Luego realiza una búsqueda exhaustiva en la vecindad para determinar un óptimo local. A medida que la búsqueda avanza se actualiza la mejor solución encontrada y se continúa hasta terminar de revisar la vecindad. Una vez que finaliza la búsqueda se hace que la mejor solución encontrada sea ahora la solución actual y se actualiza la mejor solución global. Este proceso continúa durante 50 iteraciones. Esta función corresponde a las líneas 7 a 25 de la Figura 3.1

3.4.1 GeneraciónVoraz()

Para evaluar una instancia se genera una primera solución actual aplicando un método voraz, utilizando el indicador $G_i = \frac{f_i}{b_i}$ asociado al proveedor i para realizar la selección de proveedores. Esta selección, se realiza eligiendo primero a los proveedores que tiene un menor valor de G_i e integrándolos a la solución actual, hasta que la capacidad de los proveedores seleccionados satisfaga la demanda

máxima al considerar los 27 escenarios. Como se observa, este mecanismo da prioridad a los proveedores que tienen un menor costo fijo asociado y una mayor capacidad de producción o abastecimiento. La Tabla 3.9, describe las funciones utilizadas por la función GeneraciónVoraz() .

Tabla 3.9. Funciones utilizadas por la función GeneraciónVoraz().

Nombre	**Descripción**
Initialize	Su objetivo es generar la primera solución actual aplicando un método voraz.
select_min	Su objetivo es seleccionar un proveedor no utilizado en la solución actual con base en el valor de G_i .

3.4.1.1 Función Initialize()

En la Figura 3.2 se muestra el algoritmo de la función Initialize(). Primero, calcula la demanda máxima, es decir se suman las demandas de todas las plantas por cada escenario, y se determina la mayor demanda de todos los escenarios (línea 2). Después, se prepara la solución actual, haciendo que ningún proveedor este seleccionado (líneas 3, 4 y 5). Luego, comienza un ciclo de selección de proveedores, el cual inicia desde el primer proveedor y finaliza cuando la suma de las capacidades de los proveedores seleccionados satisfaga la demanda máxima. Conforme se van seleccionando los proveedores se va actualizando la solución actual, así como su capacidad conjunta (línea 6, 7, 8, y 9). La función select_min() es invocada para determinar el proveedor no seleccionado que tiene el menor valor de G_i (línea 7).

Algoritmo: Initialize

Objetivo: Genera la primera solución actual aplicando un método voraz.

Entrada: Datos de la instancia a evaluar

Salida: solución _ actual

Figura 3.2 Algoritmo de la función Initialize()

1	Capacidad_conjunta=0
2	$Demanda\_Maxima = \max_{s \in S}(\sum_{j \in N} d_{js})$
3	Para i =1 $\rightarrow$ Num_Proveedores
4	solución _ actual[i]=0
5	Fin para i
6	Mientras Capacidad_conjunta < Demanda_Maxima
7	i = select_min()
8	Capacidad_conjunta=Capacidad_conjunta+Capacidad[i]
9	Fin Mientras

Figura 3.2 Continuación …

3.4.1.2 Función select_min()

La Figura 3.3 muestra el algoritmo de la función select_min(). Como se puede observar, primero se preparan los contadores y acumuladores a utilizar. La variable G_i_Min se utiliza para almacenar el menor valor de G_i y al principio se inicializa con un valor muy grande (líneas 1 y 2). Después se inicia un ciclo para analizar todos los proveedores (líneas 3 a 10). En la iteración i se verifica si el proveedor i no esta seleccionado en la solución actual (línea 4). Si esto ocurre se verifica si su valor de G_i es menor que G_i_Min (línea 5). Si esto ocurre se actualizan la variable G_i_Min con el valor de G_i y la variable Indice_proveedor con el valor de i de esta manera se registra el índice del proveedor que cumple con estas condiciones (líneas 6 y 7). Al término del ciclo se selecciona en la solución actual el proveedor registrado en la variable Indice_proveedor (línea 11) y la función finaliza regresando el valor de dicha variable.

Función select_min()

Objetivo: Seleccionar un proveedor no utilizado en la solución actual y que tenga la menor G_i.

Entrada: Datos de la instancia a evaluar y la solución_inicial.

Salida: Selección del proveedor que cumpla con dicho objetivo.

```
1   G_i_Min =1000000
2   Indice_proveedor=0
3   Para i =1 → Num_Proveedores
4       Si solución_inicial[i]==0
5           Si G_i < G_i_Min
6               G_i ← G_i_Min
7               indice_proveedor ← i
8           Fín Si
9       Fín Si
10  Fín Para
11  Solución_inicial[Indice_proveedor]=1
```

Figura 3.3 Algoritmo de la función select_min()

3.4.2 Función BúsquedaLocal()

A partir de la solución actual construye una vecindad de soluciones con base en el valor esperado de los precios sombra asociados a la solución inicial. Luego realiza una búsqueda exhaustiva en la vecindad para determinar un óptimo local. A medida que la búsqueda avanza se actualiza la mejor solución encontrada y se continúa hasta terminar de revisar la vecindad. El procedimiento continúa durante 50 iteraciones. La Tabla 3.10 describen dos listas que son utilizadas por la función BúsquedaLocal(). Ambas listas son de tamaño de $nSols$, donde $nSols = 2^{num\_proveedores}$. La Tabla 3.11 describe las funciones que son utilizadas por la función BúsquedaLocal().

Tabla 3.10 Listas globales que permiten mantener un registro histórico de las soluciones evaluadas en el proceso.

Nombre	Descripción
*coded_sol	Contiene la representación decimal ($H(y)$) de las soluciones evaluadas.
*coded_value	Contiene el costo de todas las soluciones evaluadas.

Tabla 3.11 Funciones utilizadas en la función BúsquedaLocal()

Nombre	Descripción
form_list()	Genera las listas de movimientos de inserción, eliminación e intercambio de proveedores que se pueden realizar sobre la solución actual para generar soluciones vecinas.
select_move()	Determina que movimiento de las listas de inserción, eliminación e intercambio, produce la mejor solución vecina. Regresa el tipo de operador y los proveedores utilizados para la construcción de la mejor solución vecina encontrada. En el proceso no se almacenan todas las soluciones vecinas generadas, simplemente se elige un movimiento de las listas, se aplica sobre la solución actual, se verifica si se debe actualizar la mejor solución vecina encontrada y al término se restaura la solución actual.
execute_move()	Recibe el operador y los proveedores utilizados para generar la mejor solución vecina encontrada. Genera una nueva solución actual aplicando, a la solución actual, el operador sobre los proveedores recibidos. Registra en las listas tabú el movimiento realizado.

En la Figura 3.4 se muestra el algoritmo de la función BúsquedaLocal(). Inicia definiendo el valor del número máximo de iteraciones a efectuar (línea 1). Posteriormente inicia un ciclo que va desde 1 hasta el número máximo de iteraciones (líneas 2 a 6). En el ciclo se invoca la función form_list() para generar las

listas de movimientos de inserción, eliminación e intercambio de proveedores (línea 3). Se invoca la función select_move(), la cual se encarga de extraer un movimiento y ejecutarlo sobre la solución actual, para obtener así, una nueva selección de proveedores o solución vecina, evaluarla y actualizar la mejor solución encontrada (línea 4). La función realiza este proceso, hasta que se agotan todos los movimientos de las tres listas de movimientos. Finalmente, la función execute_move() es invocada, para hacer tabú el movimiento ejecutado sobre la solución actual; dicho movimiento es el que permite generar la mejor solución encontrada (línea 5). Al terminar el ciclo se reporta la mejor solución encontrada.

Algoritmo: Búsqueda Local

Objetivo: Realiza una búsqueda exhaustiva en la vecindad de la solución actual para determinar un óptimo local.

Entrada: Datos de la instancia a evaluar y la solución_inicial

Salida: Selección de la mejor solución de la instancia analizada.

```
1       Num_Iteraciones ← 50
2       Para  i =1 →  Num_Iteraciones
3               Form_list
4               select_move
5               execute_move
6       Fín Para
```

Figura 3.4 Algoritmo de la función BusquedaLocal ()

3.4.2.1 Función form_list()

La Tabla 3.12 describe las listas de movimientos de inserción, eliminación e intercambio que se utilizan en la función form_list(). En la Tabla 3.13 se muestran las tres listas tabú que son utilizadas por esta función.

Tabla 3.12 Listas de movimientos de inserción, eliminación e intercambio.

Nombre	**Descripción**
int *Ins_List	Vector que contiene la lista de los 3 proveedores de menor costo relativo que se pueden insertar en la solución actual.
int *Del_List	Vector que contienen la lista de los 3 proveedores de mayor costo relativo que se pueden eliminar de la solución actual.
int **Swap_List	Matriz binaria en la que se registran los movimientos de intercambio de proveedores de menor costo relativo que se pueden realizar en la solución actual. El costo relativo de un movimiento (i, j), para el cual el proveedor i esta seleccionado y el proveedor j no esta seleccionado, se define como la diferencia $r_j - r_i$. El número máximo de movimientos que se registran en la matriz esta dado por $\frac{m^2 - m}{8}$

Tabla 3.13 Listas tabú de inserción, eliminación e intercambio.

Nombre	**Descripción**
int *ins_tabu	Lista en la que se registra el proveedor que se eliminó de la solución actual para obtener la mejor solución encontrada. Durante un número de iteraciones igual a $\frac{m}{3}$, los proveedores registrados en ésta lista, no pueden ser insertados en la solución actual para generar soluciones vecinas.

Tabla 3.13 Continuación.

Nombre	**Descripción**
int *del_tabu	Lista en la que se registra el proveedor que se insertó en la solución actual para obtener la mejor solución encontrada. Durante un número de iteraciones igual a $\frac{m}{3}$, los proveedores registrados en ésta lista, no pueden ser eliminados de la solución actual para generar soluciones vecinas.
int **swap_tabu	Matriz binaria en la que se registra el movimiento de intercambio que se realizó en la solución actual para obtener la mejor solución encontrada. Durante un número de iteraciones igual a $\frac{m(m-1)}{16}$, los movimientos registrados en ésta lista, no pueden ser realizados en la solución actual para generar soluciones vecinas.

La Figura 3.5 contiene el algoritmo de la función form_list. El algoritmo inicia con un ciclo en el que se determinan los costos relativos de todos los proveedores (líneas 1 a 7). En el ciclo se calcula el valor esperado de los precios sombra del proveedor actual correspondientes a los 27 escenarios (líneas 3, 4 y 5) y este se utiliza para calcular el costo relativo del proveedor actual (línea 6).

En este punto se asigna a la variable r_i_mejor un valor muy grande y a la variable $indice\_proveedor\_ins$ el valor CERO (línea 7).

A continuación, se inicia un ciclo general para determinar la lista de movimientos de inserción, el cual realiza $SizeIns$ iteraciones (líneas 8 a 22). Dentro del ciclo se inicia un ciclo interno en el que se analizan todos los proveedores (líneas 9 a 19). Este ciclo, se inicia determinando los proveedores no seleccionados en la solución inicial (línea 10) y se verifica que no estén registrados en la lista de inserción (línea 11) y que no este registrado en la lista tabú de inserción (línea 12).

Si el proveedor cumple las condiciones anteriores y es el de menor costo relativo (línea 13), se registran el índice del proveedor en la variable $indice\_proveedor\_ins$ y su costo relativo en la variable r_i_mejor (líneas 14 y 15). Una vez que termina el ciclo interno, se incorpora a la lista de movimientos de inserción el proveedor registrado en $indice\_proveedor\_ins$ (línea 20).

En este punto, se asigna a la variable r_i_mejor un valor muy pequeño y a la variable $indice\_proveedor\_del$ el valor CERO (línea 23).

En seguida, se inicia otro ciclo general para determinar la lista de movimientos de eliminación, el cual realiza $SizeDel$ iteraciones (líneas 24 a 40). Dentro de este ciclo se inicia otro ciclo interno para analizar todos los proveedores (líneas 25 a 38). Se determina si el proveedor actual esta seleccionado en la solución actual (línea 26) y se verifica que aún cuando se elimine el proveedor actual de la solución actual, la solución vecina resultante satisface la demanda máxima de la instancia (línea 27). Si se cumple la condición anterior, se verifica que el proveedor actual no este registrado en la lista de eliminación (línea 28). Si esto ocurre, se verifica que el proveedor actual no este registrado en la lista tabú de inserción (línea 29). Finalmente, se determina si el proveedor es el de mayor costo relativo (línea 30) y se registra el índice del proveedor $indice\_proveedor\_del$ y su costo relativo en r_i_mejor (líneas 31 y 32). Al término del ciclo se registra en lista de eliminación al proveedor especificado por $indice\_proveedor\_del$ (línea 39).

En este punto se asigna a la variable r_i_mejor un valor muy grande y a las variables $indice\_proveedor\_ins$ y $indice\_proveedor\_del$ el valor de CERO (línea 41).

A continuación, se inicia un tercer ciclo general para determinar la lista de movimientos de intercambio, el cual realiza $SizeSwap$ iteraciones (líneas 42 a 61). Dentro de este ciclo general se inicia un ciclo interno para analizar todos los proveedores (líneas 43 a 59). Se determinan todas las posibles parejas de proveedores que se pueden intercambiar en la solución actual (líneas 45 a 57). Para

cada pareja de proveedores que se pueden intercambiar se verifica que no este registrada en la lista de movimientos de intercambio (línea 47). Si esto ocurre, se verifica que la pareja no esta registrada en la lista tabú de intercambio (línea 48) y que el movimiento correspondiente a la pareja sea el de menor costo relativo de intercambio (línea 49). Finalmente, se registra el índice del proveedor que se va a insertar en $indice\_proveedor\_ins$, el índice del proveedor que se va a eliminar en $indice\_proveedor\_del$ y el costo relativo del movimiento de intercambio de la pareja en r_i_mejor (líneas 50, 51 y 52). Cuando el ciclo interno termina, se registra en la lista de movimientos de intercambio, la pareja de proveedores ($indice\_proveedor\_ins, indice\_proveedor\_del$) (línea 60).

Algoritmo: form_list

Objetivo: Generar una vecindad a partir de una solución inicial

Entrada: Datos de la instancia, Estructuras de las listas tabú, Estructura del mejor movimiento, las listas globales coded_sol y coded_value, la solución inicial y la capacidad de la solución inicial.

Salida: La vecindad, de la cual después se extraerá el conjunto elite

1 Para i =1 $\rightarrow$ Num_Proveedores

2 Para s =1 $\rightarrow$ Num_Escenarios

3 $$E(®_i) = \sum_{s \in S} p_s ®_{is}$$

4 Fín Para

5 $$r_i = \begin{cases} \frac{E(®_i)}{f_i} & si \quad E(®_i) < 0 \\ f_i & si \quad E(®_i) = 0 \end{cases}$$

6 Fín Para

Figura 3.5 Algoritmo de la función form_list ()

```
7   r_i _ mejor ← 1000000
    indice _ proveedor _ ins ← 0
8   Para cuantos =1 → SizeIns
9       Para i =1 → Num_Proveedores
10              Si solución_inicial[i]=0
11                  Si i ∉ Ins _ List
12                      Si ins_tabu[i]=0
13                              Si r_i < r_i _ mejor
14                                  r_i _ mejor ← r_i
15                                  indice _ proveedor _ ins ← i
16                              End Si
17                      End Si
18                  End Si
19              End Si
20      Fín Para
21      Ins_List = Ins_List ∪ indice_proveedor _ ins
22  Fín Para
23  r_i _ mejor ← −1000000
    indice _ proveedor _ del ← 0
24  Para cuantos =1 → SizeDel
25      Para i =1 → Num_Proveedores
26          Si solución_inicial[i]=1
27              Si capacidad_solucion_inicial - capacidad[i] >=
                Demanda_Maxima
28                  Si i ∉ Del _ List
29                      Si del_tabu[i]=0
```

Figura 3.5 Continuación …

```
30                                  Si r_i > r_i _ mejor
31                                      r_i _ mejor ← r_i
32                                      indice _ proveedor _ del ← i
33                                  End Si
34                              End Si
35                          End Si
36                      End Si
37                  End Si
38              Fín Para
39              Del_List = Del_List ∪ indice_proveedor _ del
40      Fín Para
        r_i _ mejor ← 1000000
41      indice _ proveedor _ ins ← 0
        indice _ proveedor _ del ← 0
42      Para cuantos =1 → SizeSwap
43          Para i =1 → Num_Proveedores
44              Si solución_inicial[i]=1
45                  Para j =1 → Num_Proveedores
46                      Si solución_inicial[j]=0
47                          Si Swap _ List[i][ j] = 0
48                              Si swap_tabu[i][j]=0
49                                  Si r_j − r_i < r_i _ mejor
50                                      r_i _ mejor ← r_j − r_i
51                                      indice _ proveedor _ ins ← i
52                                      indice _ proveedor _ del ← j
53                                  End Si
```

Figura 3.5 Continuación …

54	End Si
55	End Si
56	End Si
57	Fín Para
58	End Si
59	Fín Para
60	$Swap\_List[cuantos][1] = indice\_proveedor\_ins$ $Swap\_List[cuantos][2] = indice\_proveedor\_del$
61	Fín Para

Figura 3.5 Continuación …

3.4.2.2 Función select_move()

La Figura 3.6 contiene el algoritmo de la función select_move(). Como se puede observar inicia determinando cuantos elementos tiene cada una de las listas de movimientos y asigna un valor muy grande a la variable $mejor\_solucion\_encontrada$ (línea 1).

En este punto inicia un ciclo general para explorar los movimientos de la lista de inserción (línea 2 a 26). Se asigna a la variable $indice\_proveedor\_ins$ el movimiento actual de la lista de inserción (línea 3). Se ejecuta el movimiento de inserción actual seleccionando en la solución actual el proveedor especificado por $indice\_proveedor\_ins$ (línea 4). Se invoca la función get_hcode() para determinar la representación decimal $H(y)$ de la solución vecina generada (línea 5). Se revisa el registro histórico para verificar si la solución vecina ya fue evaluada anteriormente (línea 6), en tal caso se recupera el valor de la función objetivo que le corresponde (línea 7). Si el valor de la solución vecina es mejor que el de la mejor solución encontrada (línea 8), entonces se verifica si el movimiento de inserción, que se utilizó para generar la solución vecina, no esta registrado en la lista tabú de inserción o si dicha solución mejora la mejor solución global (línea 9). Si alguna de las condiciones anteriores se cumple, se registran el tipo de operador utilizado para

generar esta solución (óptima local o global), el proveedor insertado y el valor de la solución (líneas 10, 11 y 12). Pero si la solución vecina no ha sido evaluada, se evalúa y se actualiza el registro histórico (líneas 15, 16 y 17). Si el valor de la solución vecina es mejor que el de la mejor solución encontrada (línea 18), entonces se verifica si el movimiento de inserción que se utilizó para generar la solución vecina no este registrado en la lista tabú de inserción o si dicha solución mejora la mejor solución global (línea 19). Si alguna de las condiciones anteriores se cumple, se registran el tipo de operador utilizado para generar esta solución (óptima local o global), el proveedor insertado y el valor de la solución (línea 20, 21 y 22). En este punto se restaura la solución actual a partir de la solución vecina, eliminando en ésta, el proveedor especificado por $indice\_proveedor\_ins$ (línea 25).

En este punto inicia un ciclo general para explorar los movimientos de la lista de eliminación (línea 27 a 51). Se asigna a la variable $indice\_proveedor\_del$ el movimiento actual de la lista de eliminación (línea 28). Se ejecuta el movimiento de eliminación actual eliminando en la solución actual el proveedor especificado por $indice\_proveedor\_del$ (línea 29). Se invoca la función get_hcode() para determinar la representación decimal $H(y)$ de la solución vecina generada (línea 30). Se revisa el registro histórico para verificar si la solución vecina ya fue evaluada anteriormente (línea 31), en tal caso se recupera el valor de la función objetivo que le corresponde (línea 32). Si el valor de la solución vecina es mejor que el de la mejor solución encontrada (línea 33), entonces se verifica si el movimiento de eliminación, que se utilizó para generar la solución vecina, no esta registrado en la lista tabú de eliminación o si dicha solución mejora la mejor solución global (línea 34). Si alguna de las condiciones anteriores se cumple, se registran el tipo de operador utilizado para generar esta solución (óptima local o global), el proveedor eliminado y el valor de la solución (línea 35, 36 y 37). Pero si la solución vecina no ha sido evaluada, se evalúa y se actualiza el registro histórico (líneas 40, 41 y 42). Si el valor de la solución vecina es mejor que el de la mejor solución encontrada (línea

43), entonces se verifica si el movimiento de eliminación que se utilizó para generar la solución vecina no este registrado en la lista tabú de eliminación o si dicha solución mejora la mejor solución global (línea 44). Si alguna de las condiciones anteriores se cumple, se registran el tipo de operador utilizado para generar esta solución (óptima local o global), el proveedor eliminado y el valor de la solución (línea 45, 46 y 47). En este punto se restaura la solución actual a partir de la solución vecina, insertando en ésta, el proveedor especificado por $indice\_proveedor\_del$ (línea 50).

En este punto inicia un ciclo general para analizar los movimientos de la lista de intercambio (línea 52 a 81). Se asigna el primero y segundo proveedor de la pareja de intercambio actual a las variables $indice\_proveedor\_del$ y $indice\_proveedor\_ins$ respectivamente (líneas 53 y 54). Se realiza el intercambio de la pareja de proveedores en la solución actual para generar una solución vecina (líneas 55 y 56). Se invoca la función get_hcode() para determinar la representación decimal $H(y)$ de la solución vecina generada (línea 57). Se revisa el registro histórico para verificar si la solución vecina ya fue evaluada anteriormente (línea 58), en tal caso se recupera el valor de la función objetivo que le corresponde (línea 59). Si el valor de la solución vecina es mejor que el de la mejor solución encontrada (línea 60), entonces se verifica si el movimiento de intercambio, que se utilizó para generar la solución vecina, no esta registrado en la lista tabú de intercambio o si dicha solución mejora la mejor solución global (línea 61). Si alguna de las condiciones anteriores se cumple, se registra el tipo de operador utilizado para generar esta solución (óptima local o global), los índices de los proveedores intercambiados y el valor de la solución (línea 62, 63, 64 y 65). Pero si la solución vecina no ha sido evaluada, se evalúa y se actualiza el registro histórico (líneas 68,69 y 70). Si el valor de la solución vecina es mejor que el de la mejor solución encontrada (línea 71), entonces se verifica si el movimiento de eliminación que se utilizó para generar la solución vecina no este registrado en la lista tabú de

eliminación o si dicha solución mejora la mejor solución global (línea 72). Si alguna de las condiciones anteriores se cumple, se registran el tipo de operador utilizado para generar esta solución (óptima local o global), los índices de los proveedores intercambiados y el valor de la solución (línea 73, 74 y 75). En este punto se restaura la solución actual a partir de la solución vecina eliminando el proveedor especificado por $indice\_proveedor\_ins$ e insertando en ésta, el proveedor especificado por $indice\_proveedor\_del$ (líneas 79 y 80).

Algoritmo: *select_move*

Objetivo: Su objetivo es evaluar el conjunto elite y seleccionar el mejor movimiento.

Entrada: Datos de la instancia, Estructuras de las listas tabú, Estructura del mejor movimiento, las listas globales coded_sol y coded_value, la solución inicial.

Salida: Selección de la mejor solución del conjunto elite

```
1   SizeIns = |Ins_List|
    SizeDel = |Del_List|
    SizeSwap = |Swap_List|
    mejor_solucion_elite ← 1000000
2   Para i =1 ì SizeIns
3       indice_proveedor_ins ← Ins_Lista[i]
4       Solución_inicial[indice_proveedor_ins]=1
5       Codigo_solucion_inicial= get_hcode(solución_inicial)
6       Si Codigo_solucion_inicial ∈ coded_sol
7           Costo_Solucion = coded_value[Codigo_solucion_inicial]
8           Si Costo_Solucion < Costo_mejor_solucion_encontrada
9               Si ( Costo_Solucion < Costo_mejor_solucion_global ) ó
                   ( ins_tabu[indice_proveedor_ins]=0)
10  Costo_mejor_solucion_encontrada ← Costo_Solucion
11                  Operador ← Inserción
12                  indice_proveedor ← indice_proveedor_ins
13              End Si
14          End Si
```

Figura 3.6 Algoritmo *select_move*

```
15  De lo contrario
16  Costo_Solucion ← Evaluar[Codigo_solucion_inicial]
17  coded_value[Codigo_solucion_inicial] = Costo_solucion
18        Si Costo_Solucion < Costo_mejor_solucion_encontrada
19           Si (Costo_Solucion < Costo_mejor_solucion_global ) ó
                ( ins_tabu[indice_proveedor_ins]=0)
20  Costo_mejor_solucion_encontrada ← Costo_Solucion
21                    Operador ← Inserción
22                    indice_proveedor ← indice_proveedor_ins
23           End Si
24        End Si
25     Solución_inicial[indice_proveedor_ins]=0
26  Fín Para
27  Para i =1 ì SizeDel
28     indice_proveedor_del ← Del_Lista[i]
29     Solución_inicial[indice_proveedor_del]=0
30     Codigo_solucion_inicial=get_hcode(solución_inicial)
31     Si Codigo_solucion_inicial ∈ coded_sol
32        Costo_Solucion = coded_value[Codigo_solucion_inicial]
33        Si Costo_Solucion < Costo_mejor_solucion_encontrada
34           Si (Costo_Solucion < Costo_mejor_solucion_global ) ó
                ( del_tabu[indice_proveedor_del]=0)
35  Costo_mejor_solucion_encontrada ← Costo_Solucion
36                    Operador ← Eliminación
37                    indice_proveedor ← indice_proveedor_del
38           End Si
39        End Si
40  De lo contrario
41  Costo_Solucion ← Evaluar[Codigo_solucion_inicial]
42  coded_value[Codigo_solucion_inicial] = Costo_solucion
43        Si Costo_Solucion < Costo_mejor_solucion_encontrada
44           Si (Costo_Solucion < mejor_solucion_global ) ó
                ( del_tabu[indice_proveedor_del]=0)
45  Costo_mejor_solucion_encontrada ← Costo_Solucion
46                    Operador ← Eliminación
47                    indice_proveedor ← indice_proveedor_del
48           End Si
49        End Si
```

Figura 3.6 Continuación …

```
50      Solución_inicial[indice_proveedor_del]=1
51  Fín Para
52  Para i =1 ì  SizeSwap
53       indice _ proveedor _ ins ← Ins _ Lista[i]
54      indice _ proveedor _ del ← Del _ Lista[i]
55      Solución_inicial[indice_proveedor_ins]=1
56      Solución_inicial[indice_proveedor_del]=0
57      Codigo_solucion_inicial=get_hcode(solución_inicial)
58      Si Codigo _ solucion _ inicial ∈ coded _ sol
59          Costo _ Solucion = coded _ value[Codigo _ solucion _ inicial]
60          Si Costo _ Solucion < Costo _ mejor _ solucion _ encontrada
              Si (Costo _ Solucion < Costo _ mejor _ solucion _ global ) ó
61          (                              swap_tabu[indice_proveedor_ins]
    [indice_proveedor_del]=0)

62      Costo _ mejor _ solucion _ encontrada ← Costo _ Solucion
63                  Operador ← Eliminación
64                  indice _ proveedor _ i ← indice _ proveedor _ ins
65                  indice _ proveedor _ j ← indice _ proveedor _ del
66              End Si
67          End Si
68  De lo contrario
69  Costo _ Solucion ← Evaluar[Codigo _ solucion _ inicial]
70  coded _ value[Codigo _ solucion _ inicial] = Costo _ solucion
71          Si Costo _ Solucion < Costo _ mejor _ solucion _ encontrada
              Si (Costo _ Solucion < mejor _ solucion _ global ) ó
72          (swap_tabu[indice_proveedor_ins]
    [indice_proveedor_del]=0)

73      Costo _ mejor _ solucion _ encontrada ← Costo _ Solucion
74                  Operador ← Eliminación
75                   indice _ proveedor _ i ← indice _ proveedor _ ins
76                  indice _ proveedor _ j ← indice _ proveedor _ del
77              End Si
78          End Si
79      Solución_inicial[indice_proveedor_ins]=0
80      Solución_inicial[indice_proveedor_del]=1
81  Fín Para
```

Figura 3.6 Algoritmo *select_move*

3.4.2.3 Función execute_move

La Figura 3.7 muestra el algoritmo de la función execute_move(). Esta función recibe el operador y los proveedores utilizados para generar la mejor solución vecina encontrada. Genera una nueva solución actual aplicando, a la solución actual, el operador sobre los proveedores recibidos. Registra en las listas tabú el movimiento realizado. Inicia determinando si el operador recibido es de inserción, eliminación o intercambio. Si el operador es de inserción (Líneas 1 a 4), se inserta el proveedor recibido en la solución actual (línea 2), se determina la capacidad conjunta de la nueva solución actual (línea 3) y se actualiza la lista tabú de eliminación (línea 4). El operador es de eliminación (Líneas 5 a 8), se elimina el proveedor recibido en la solución actual (línea 6), se determina la capacidad conjunta de la nueva solución actual (línea 7) y se actualiza la lista tabú inserción (línea 8). Si el operador es de intercambio (Líneas 9 a 14), se realiza el intercambio de los proveedores recibidos en la solución actual (líneas 10 y 11), se determina la capacidad conjunta de la nueva solución actual (línea 12) y se actualiza la lista tabú de intercambio (línea 13). En este punto, la solución actual es la mejor solución encontrada en la búsqueda exhaustiva realizada por la función select_move(), por lo que se determina si esta solución es mejor que la mejor solución global (línea 15). En tal caso, se actualiza la mejor solución global (línea 16).

Algoritmo: execute_move

Objetivo: Su objetivo es actualizar las listas tabú, en base al mejor movimiento seleccionado.

Entrada: Datos de la instancia a evaluar, listas tabú, mejor moviendo, datos de la solución actual

Salida: Actualizar las listas tabú y actualizar la mejor solución global conocida

Figura 3.7 Algoritmo execute_move

```
1    Si Operador = Inserción
2        Solución_inicial[indice_proveedor]=1
3        Capacidad_conjunta=
         Capacidad_conjunta + Capacidad[indice_proveedor]
4        Del_Tabu[indice_proveedor] = duracion_tabu
5    Pero Si Operador = Eliminación
6        Solución_inicial[indice_proveedor]=0
7        Capacidad_conjunta=
         Capacidad_conjunta - Capacidad[indice_proveedor]
8
     Ins_Tabu[indice_proveedor] = duracion_tabu
9    Pero Si Operador = Intercambio
10       Solución_inicial[indice_proveedor_ins]=1
11       Solución_inicial[indice_proveedor_del]=0
         Capacidad_conjunta= Capacidad_conjunta +
12           Capacidad[indice_proveedor_ins] -
             Capacidad[indice_proveedor_del]
13
     Swap_Tabu[indice_proveedor_ins][indice_proveedor_del] = duracion_tabu
14   End Si
15   Si mejor_solucion_encontrada < mejor_solucion_global
16           mejor_solucion_global ← mejor_solucion_encontrada
17   End Si
```

Figura 3.7 Continuación …

Capítulo 4

PROPUESTA DE SOLUCIÓN

En este capitulo se presentan las estrategias propuestas, así como también la estrategia de referencia. Se documenta detalladamente el pseudocódigo de los algoritmos asociados con cada estrategia.

4.1 Enfoque de solución

Las soluciones reportadas del problema ROCIS consideran básicamente dos estrategias de prioridades para elegir los proveedores que se incorporan a la solución inicial. La primera consiste en dar prioridad a los proveedores de menor costo fijo y mayor capacidad de producción y la segunda incorpora a los elementos anteriores el valor esperado del costo de enviar productos del proveedor a todas las plantas.

La principal limitación de la primera alternativa es que no incorpora el costo de envío y aún cuando este factor es incorporado por la segunda estrategia, el mecanismo que se utiliza parece demasiado pesimista.

En este trabajo se propone modificar el mecanismo de incorporación del costo de envío, para que se consideren únicamente las plantas hacia las que resulta más económico el envío de los productos desde el sitio del proveedor. Para validar este enfoque se proponen dos modelos alternativos de este concepto.

4.2 Estrategias de prioridades consideradas

En este capitulo se describen las estrategias de prioridades que se consideran en este trabajo. Las estrategias E_1 y E_2 son las reportadas en [González 2004] y [González 2006] respectivamente, la estrategia E_3 corresponde a un modelo que se basa en una línea abierta propuesta en [González 2006], las estrategias E_4 y E_5 corresponden a dos nuevos modelos propuestos del concepto de las plantas más económicas con respecto al costo de envío de productos desde el sitio del proveedor.

Las estrategias E_1, E_2, E_4 y E_5 construyen una buena solución inicial, construyen una vecindad y realizan una búsqueda exhaustiva en la vecindad. La estrategia E_3 esta compuesta de dos fases, la primera fase consiste en construir una buena solución inicial, la segunda fase consiste en utilizar dicha solución inicial para localizar las plantas frecuentemente servidas y con esa información refinar la solución inicial, después con esa nueva solución inicial se construye una vecindad para realizar dentro de la misma una búsqueda local. Las estrategias E_2, E_3, E_4 y E_5

son básicamente modificaciones de la estrategia E_1. A continuación se describe cada una de estas estrategias.

4.2.1. Estrategia E_1: proveedores de menor costo fijo y mayor capacidad de producción.

Para cada proveedor i se calcula en base a $G_i = \frac{f_i}{b_i}$, se ordenan ascendentemente los proveedores de acuerdo a G_i y se van eligiendo progresivamente hasta que se construya una solución inicial que satisfaga la demanda requerida en todos los escenarios. La estrategia de prioridades para elegir los proveedores que se integran a la solución actual se define en la formulación de G_i. En este caso la estrategia consiste en elegir primero a los proveedores de menor costo fijo y mayor capacidad de producción. Esta es la estrategia utilizada en la solución propuesta en [González 2004].

En la tabla 4.1, se muestra el pseudocódigo del algoritmo para generar la solución inicial con la estrategia E_1.

Tabla 4.1 Construcción de la solución inicial con la estrategia E_1

Línea	Descripción
1	Generar la solución inicial y
2	Calcular $D = \max_{s \in S}(\sum_{j \in N} d_{js})$
3	Ordenar los proveedores de manera ascendente por $G_i = \frac{f_i}{b_i}$
4	Seleccionar los proveedores iniciando con el primero de la lista ordenada hasta que la suma de las capacidades de los proveedores seleccionados sea mayor que D.
5	Determinar $F(y)$, resolviendo los subproblemas de distribución que se generan en todos los escenarios.

4.2.2. Proveedores de menor costo fijo, mayor capacidad de producción y menor valor esperado del costo de enviar a todas las plantas (E_2).

Esta estrategia es la que se utiliza en el método de solución propuesto en [González 2006]. Se implementa definiendo:

$$G_i = \frac{f_i + \left(\sum_{s \in S} p_s \left(\sum_{j=1}^{n} c_{ij} e_{is} \right) \right)}{b_i}, \qquad \forall i \in M \tag{4.1}$$

En la tabla 4.2, se muestra el pseudocódigo del algoritmo para generar la solución inicial con la estrategia E_2.

Tabla 4.2 Construcción de solución inicial con la estrategia E_2

Línea	Descripción
1	Generar la solución inicial y
2	Calcular $D = \max_{s \in S} (\sum_{j \in N} d_{js})$
3	Ordenar los proveedores de manera ascendente por $G_i = \frac{f_i + \left(\sum_{s \in S} p_s \left(\sum_{j=1}^{n} c_{ij} e_{is} \right) \right)}{b_i}, \qquad \forall i \in M$
4	Seleccionar los proveedores iniciando con el primero de la lista ordenada hasta que la suma de las capacidades de los proveedores seleccionados sea mayor que D.
5	Determinar $F(y)$, resolviendo los subproblemas de distribución que se generan en todos los escenarios.

4.2.3. Proveedores de menor costo fijo, mayor capacidad de producción y menor valor esperado del costo de enviar a las plantas frecuentemente servidas, refinando la solución inicial previa a la búsqueda local (E_3).

Dada una selección de proveedores $y = [y_1, y_2, .., y_m]$, para la cual ya se ha calculado el valor de la función objetivo $F(y)$, entonces el número de servicios proporcionados

por el proveedor i a la planta j en los diferentes escenarios considerados, esta dado por:

$$s_{ij} = \sum_{s \in S} xb_{ijs}$$
$$donde$$
$$xb_{ijs} = \begin{cases} 1 & si \quad x_{ijs} > 0 \\ 0 & si \quad x_{ijs} = 0 \end{cases} \tag{4.2}$$

El conjunto de plantas frecuentemente servidas por el proveedor i se define entonces como:

$$P_i^+ = \left\{ j \in N \,\middle|\, s_{ij} > SE_i \right\} \qquad \forall i \in M$$
$$donde$$
$$SE_i = \frac{\sum_{j=1}^{n} s_{ij}}{n} \tag{4.3}$$

En la tabla 4.3, se muestra el pseudocódigo del algoritmo para generar la solución inicial con la estrategia E_3.

Tabla 4.3 Construcción de solución inicial con la estrategia E_3

Fase I	
1	Generar la solución inicial y
2	Calcular $D = \max_{s \in S}(\sum_{j \in N} d_{js})$
3	Ordenar los proveedores de manera ascendente por $G_i = \frac{f_i}{b_i}$
4	Seleccionar los proveedores iniciando con el primero de la lista ordenada hasta que la suma de las capacidades de los proveedores seleccionados sea mayor que D.
5	Determinar $F(y)$, resolviendo los subproblemas de distribución que se generan en todos los escenarios.

Tabla 4.3 Continuación …

Fase II	
6	A partir de las soluciones vinculadas a la solución inicial:
7	Determinar el número de servicios el proveedor i a la planta j $s_{ij} = \sum_{s \in S} xb_{ijs}$ *donde* $xb_{ijs} = \begin{cases} 1 & si \quad x_{ijs} > 0 \\ 0 & si \quad x_{ijs} = 0 \end{cases}$
8	Determinar el servicio esperado de cada proveedor $SE_i = \frac{\sum_{j=1}^{n} s_{ij}}{n}$
9	Determinar el conjunto de plantas frecuentemente servidas por el proveedor i $P_i^+ = \left\{ j \in N \middle\| s_{ij} > \sim (SE_i) \right\} \qquad \forall i \in M,$
10	Ordenar los proveedores de manera ascendente por $G(i) = \frac{f_i + \left(\sum_{s \in S} p_s \left(\sum_{j \in P_i^+} c_{ij} e_{is} \right) \right)}{b_i}$
	Construir la selección de proveedores y iniciando con el primero de la lista ordenada hasta que la suma de las capacidades de los proveedores seleccionados sea mayor que la demanda máxima D.
11	Determinar $F(y)$, resolviendo los subproblemas de distribución que se generan en todos los escenarios.

4.2.4. Proveedores de menor costo fijo, mayor capacidad de producción y menor valor esperado del costo de enviar a las plantas de menor costo (E_4 y E_5).

Como se señaló las estrategias E_2 y E_3 parecen ser demasiado pesimistas, por lo que se propone limitar el número de plantas que se deben considerar para calcular el valor de G_i. En términos prácticos parece ser más razonable incorporar en el indicador únicamente las plantas hacia las que el envío es más económico. Para redefinir el indicador sea $C_i = \{C_{ij}, j = 1,2,...,n\}$ el conjunto de costos de envío del proveedor i a todas las plantas. Luego si es posible definir un costo frontera CF_i considerando únicamente los costos incluidos en C_i, el conjunto de plantas hacia las que al proveedor i le resulta más económico enviar producto es:

$$P_i^+ = \{ j \in N \mid c_{ij} < CF_i \} \tag{4.4}$$

Existen diversas alternativas para modelar el costo frontera CF_i considerando únicamente los elementos de C_i. Una primera posibilidad es modelarlo como el promedio de los costos de envió a todas las plantas. En este caso, el costo frontera se define como:

$$CF1_i = \frac{\sum_{j=1}^{n} C_{ij}}{n}, \forall i \in M \tag{4.5}$$

Por otra parte, si $C_{\min}$ y $C_{\max}$ son el costo mínimo y máximo respectivamente de todos los costos de C_i, entonces otra forma de modelar el costo frontera es la siguiente:

$$CF2_i = C_{\min} + \sim (C_{\max} - C_{\min}), \quad \forall i \in M \quad donde \quad 0 < \sim < 1 \tag{4.6}$$

Para que en G_i se consideren únicamente las plantas de P^+ su definición debe ser la siguiente:

$$G(i) = \frac{f_i + \left(\sum_{s \in S} p_s \left(\sum_{j \in P_i^+} c_{ij} e_{is} \right) \right)}{b_i} \tag{4.7}$$

Las estrategias E_4 y E_5 corresponden respectivamente a los costos frontera CF_i modelados por $CF1_i$ y $CF2_i$.

En la tabla 4.4, se muestra el pseudocódigo del algoritmo para generar la solución inicial con la estrategia E_4.

Tabla 4.4 Construcción de solución inicial con la estrategia E_4

Línea	**Descripción**
1	Generar la solución inicial factible y
2	Calcular $D = \max_{s \in S} (\sum_{j \in N} d_{js})$
3	Calcular el costo de envió de cada proveedor a todas las plantas $C_i = \{C_{ij}, j = 1, 2, .., N\}$
4	Se calcula el costo de frontera para todos los proveedores $CF_i = \frac{\sum_{j=1}^{n} C_{ij}}{n}, \forall i \in M$
5	Calcular el conjunto de plantas hacia las que al proveedor i le resulta más económico enviar producto $P_i^+ = \{ j \in N \mid c_{ij} < CF_i \}$.
6	Ordenar los proveedores de manera ascendente por $G(i) = \frac{f_i + \left(\sum_{s \in S} p_s \left(\sum_{j \in P_i^+} c_{ij} e_{is} \right) \right)}{b_i}$

Tabla 4.4 Continuación …

Línea	Descripción
7	Seleccionar los proveedores iniciando con el primero de la lista ordenada hasta que la suma de las capacidades de los proveedores seleccionados sea mayor que D.
8	Determinar $F(y)$, resolviendo los subproblemas de distribución que se generan en todos los escenarios.

En la tabla 4.5, se muestra el pseudocódigo del algoritmo para generar la solución inicial con la estrategia E_5.

Tabla 4.5 Construcción de solución inicial con la estrategia E_5

Línea	Descripción
1	Generar la solución inicial factible y
2	Calcular $D = \max_{s \in S}(\sum_{j \in N} d_{js})$
3	Calcular el costo de envió de cada proveedor a todas las plantas $C_i = \{C_{ij},\ j = 1,2,..,N\}$
4	Calcular los costos mínimos y máximos de todos los proveedores y se calculan sus respectivos costos de frontera $CF_i = C_{\min} + \sim (C_{\max} - C_{\min}), \quad \forall i \in M \quad donde \quad 0 < \sim < 1$
5	Calcular el conjunto de plantas hacia las que al proveedor i le resulta más económico enviar producto $P_i^+ = \{ j \in N \mid c_{ij} < CF_i \}$.
6	Ordenar los proveedores de manera ascendente por $G(i) = \dfrac{f_i + \left(\sum_{s \in S} p_s \left(\sum_{j \in P_i^+} c_{ij} e_{is} \right) \right)}{b_i}$

Tabla 4.5 Continuación …

Línea	Descripción
7	Seleccionar los proveedores iniciando con el primero de la lista ordenada hasta que la suma de las capacidades de los proveedores seleccionados sea mayor que D.
8	Determinar $F(y)$, resolviendo los subproblemas de distribución que se generan en todos los escenarios.

Capítulo 5

RESULTADOS EXPERIMENTALES

En este capitulo se muestran los resultados experimentales de las estrategias propuestas. Se realiza una evaluación comparativa midiendo su eficiencia y su eficacia en el desempeño.

5.1 Plataforma experimental 74
5.2 Experimento. Análisis comparativo del desempeño de las estrategias propuestas. 75
5.2.1 Objetivo 75
5.2.2 Procedimiento 75
5.2.3 Resultados 75
5.2.4 Análisis de resultados 77
5.3 Experimento 2. Análisis de la eficiencia de la estrategia de referencia respecto a la estrategia E_5 con alfa en 0.8 78
5.3.1 Objetivo 78
5.3.2 Procedimiento 78
5.3.3 Resultados 78
5.3.4 Análisis de resultados 79

Tabla 5.1 Resultados de referencia 75
Tabla 5.2 Resultados que se obtienen al utilizar la estrategia E_2 76
Tabla 5.3 Resultados que se obtienen al utilizar la estrategia E_3 76
Tabla 5.4 Resultados que se obtienen al utilizar la estrategia E_4 76
Tabla 5.5 Resultados que se obtienen al utilizar la estrategia E_5 con α=0.8 77
Tabla 5.6 Resultados comparativos de las estrategias propuestas 78
Tabla 5.7 Resultados que se obtienen al utilizar la solución de referencia con 10 iteraciones 79
Tabla 5.8 Resultados que se obtienen al utilizar la estrategia E_5 con α=0.8 con 10 iteraciones. 79

5.1 Plataforma experimental

Las estrategias se implementaron sobre el código de la solución de referencia [González 2004]. Las 90 instancias utilizadas para evaluar del desempeño de los algoritmos, son las que se reportan en [González 2004]. Todas las instancias consideran 10 plantas y 27 escenarios. Las 90 instancias se dividen en tres grupos de 30, estos tres grupos consideran 10, 15 y 20 proveedores respectivamente. Estas 90 instancias, conforman el paquete de instancias de referencia. Se determinó la solución óptima de todas las instancias utilizando un método exhaustivo.

Los experimentos se realizaron en una computadora Dell Optiplex 160L con procesador Pentium 4 a 2.4 GHZ y 1 GB de memoria RAM. El código se compiló utilizando Visual C++ 6.0 y el sistema operativo Windows XP Profesional con service pack 2. Para la solución de los subproblemas de transporte, se utilizó LINDO API versión 2.0.

Todas las tablas que muestran los resultados de la experimentación realizada, tienen las mismas características, las cuales se describen a continuación.

La primera columna de cada tabla contiene el número de proveedores del grupo analizado, la segunda muestra el número promedio de las iteraciones realizadas para llegar a la mejor solución encontrada, la tercera columna contiene el tiempo promedio en segundos de cpu utilizados para llegar a la mejor solución. La cuarta columna muestra el tiempo promedio utilizado para resolver una instancia del grupo correspondiente. La última columna muestra el número de instancias de cada uno de los grupos para las cuales se encontró la solución óptima. Las tablas también incluyen los valores acumulados de cada una de estas características.

Para evaluar el desempeño de los diferentes algoritmos, se evalúa la eficiencia y la calidad de solución de los mismos. Para evaluar la calidad de solución se utiliza como métrica el número de soluciones optimas encontradas. Para evaluar la eficiencia se utilizan las siguientes tres métricas:

- el promedio de iteraciones requeridas para encontrar la mejor solución,
- el promedio del tiempo requerido para encontrar la mejor solución, y
- el tiempo promedio requerido para la solución de una instancia.

5.2 Experimento. Análisis comparativo del desempeño de las estrategias propuestas.

5.2.1 Objetivo

Evaluar el desempeño de las estrategias propuestas con respecto al desempeño de la solución de referencia.

5.2.2 Procedimiento

Se determina el desempeño en calidad y eficiencia de la solución de referencia. Se modifica la solución de referencia para incorporar cada una de las estrategias propuestas. Se determina el desempeño de cada una de las soluciones modificadas resolviendo el paquete de instancias especificado.

5.2.3 Resultados

La Tabla 5.1 muestra el desempeño de la solución de referencia. Las tablas 5.2, 5.3, 5.4 y 5.5 muestran el desempeño de la solución modificada al incorporar las estrategias E_2, E_3, E_4 y E_5 respectivamente.

Tabla 5.1 Resultados de referencia

Proveedores	Promedio de iteraciones hasta el mejor	Tiempo promedio hasta el mejor	Tiempo promedio para resolver una instancia	Óptimos encontrados
10	3.07	2.62	10.25	30/30
15	4.77	9.13	38.17	30/30
20	5.47	20.32	92.45	28/30
Acumulados	13.30	32.07	140.87	88/90

Tabla 5.2 Resultados que se obtienen al utilizar la estrategia E_2

Proveedores	Promedio de iteraciones hasta el mejor	Tiempo promedio hasta el mejor	Tiempo promedio para resolver una instancia	Óptimos encontrados
10	2.07	1.83	9.61	30/30
15	3.80	6.87	34.99	29/30
20	6.00	19.19	93.04	29/30
Acumulados	11.87	27.89	137.64	88/90

Tabla 5.3 Resultados que se obtienen al utilizar la estrategia E_3

Proveedores	Promedio de iteraciones hasta el mejor	Tiempo promedio hasta el mejor	Tiempo promedio para resolver una instancia	Óptimos encontrados
10	3.00	2.71	6.85	29/30
15	4.60	9.35	23.26	27/30
20	5.70	21.99	57.75	27/30
Acumulados	13.30	34.05	87.86	83/90

Tabla 5.4 Resultados que se obtienen al utilizar la estrategia E_4

Proveedores	Promedio de iteraciones hasta el mejor	Tiempo promedio hasta el mejor	Tiempo promedio para resolver una instancia	Óptimos encontrados
10	3.07	2.52	9.90	30/30
15	4.77	8.83	37.05	30/30
20	5.47	19.78	90.19	28/30
Acumulados	13.30	31.13	137.13	88/90

Tabla 5.5 Resultados que se obtienen al utilizar la estrategia E_5 con α=0.8

Proveedores	Promedio de iteraciones hasta el mejor	Tiempo promedio hasta el mejor	Tiempo promedio para resolver una instancia	Óptimos encontrados
10	2.40	2.03	9.59	30/30
15	3.20	6.05	33.79	29/30
20	4.90	17.04	89.69	30/30
Acumulados	10.50	25.13	133.07	89/90

5.2.4 Análisis de resultados

La Tabla 5.6 muestra el porcentaje de incremento observado en cada una de las métricas utilizadas cuando se aplica cada una de las estrategias propuestas, con respecto al valor observado cuando se utiliza la solución de referencia. La primera columna de contiene la estrategia evaluada, las siguientes 4 columnas contienen el porcentaje de incremento observado en el promedio del número de iteraciones, el promedio del tiempo requerido para alcanzar la mejor solución, el promedio de tiempo requerido para resolver una instancia y el número de soluciones óptimas encontradas, respectivamente.

De manera general se puede observar que las estrategias E_4 y E_5 muestran claramente un mejor desempeño que las estrategias E_2 y E_3. Las estrategias E_4 y E_5 muestran el mismo desempeño con respecto a la métrica de calidad, ya que ambas logran un incremento de 1.14% en el número de óptimos encontrados con respecto a los encontrados con la solución de referencia. Sin embargo el desempeño de la estrategia E_5 con respecto a las métricas de eficiencia es claramente superior al desempeño de la estrategia E_4. Por lo tanto se considera que la estrategia E_5 es la que muestra un mejor desempeño ya que incrementa en un 1.14% la calidad de la solución y disminuye el número promedio de iteraciones para alcanzar la mejor solución, el tiempo promedio requerido para alcanzar esta solución y el tiempo

promedio requerido para evaluar cada una de las instancias en 21.05%, 19.66% y 3.32% respectivamente.

Tabla 5.6 Resultados comparativos de las estrategias propuestas

	Métricas de eficiencia			**Métrica de calidad**
1	**2**	**3**	**4**	**5**
E_2	-10.78%	-13.02%	-2.29%	0.00%
E_3	0.00%	6.18%	-37.63%	-5.68%
E_4	-23.06%	-14.88%	3.17%	1.14%
E_5 con α=0.8	-21.05%	-19.66%	-3.32%	1.14%

5.3 Experimento 2. Análisis de la eficiencia de la estrategia de referencia respecto a la estrategia E_5 con alfa en 0.8

5.3.1 Objetivo

Evaluar el desempeño de la estrategia E_5 con respecto al desempeño de la solución de referencia, al disminuir el número de iteraciones.

5.3.2 Procedimiento

Se determina el desempeño en calidad y eficiencia de la estrategia E_5 con respecto a la solución de referencia. Se fija el número de iteraciones a 10. Se determina el desempeño de cada una de las soluciones modificadas resolviendo el paquete de instancias especificado.

5.3.3 Resultados

La Tabla 5.7 muestra el desempeño de la solución de referencia con 10 iteraciones. La Tabla 5.8 muestra el desempeño de E_5 con α=0.8 con 10 iteraciones.

Tabla 5.7 Resultados que se obtienen al utilizar la solución de referencia con 10 iteraciones

Proveedores	Promedio de iteraciones hasta el mejor	Tiempo promedio hasta el mejor	Tiempo promedio para resolver una instancia	Óptimos Encontrados
10	3.07	2.51	6.43	30/30
15	3.97	7.72	17.32	28/30
20	5.27	19.46	34.42	27/30
Acumulados	12.30	29.70	58.17	85/90

Tabla 5.8 Resultados que se obtienen al utilizar la estrategia E_5 con α=0.8 con 10 iteraciones.

Proveedores	Promedio de iteraciones hasta el mejor	Tiempo promedio hasta el mejor	Tiempo promedio para resolver una instancia	Óptimos
10	2.40	2.03	6.29	30/30
15	2.77	5.36	16.73	28/30
20	3.93	14.51	32.82	28/30
Acumulados	9.10	21.90	55.83	86/90

5.3.4 Análisis de resultados

De manera general se puede observar que la estrategia E_5 con α=0.8 muestra claramente un mejor desempeño en calidad y eficiencia que la estrategia E_1, evaluadas ambas con 10 iteraciones. La estrategia E_5 con α=0.8 logra un incremento de 1.18% en el número de óptimos encontrados, además obtiene una mejora de 26.02% en el número promedio de iteraciones para alcanzar la mejor solución, así como en el tiempo promedio requerido para alcanzar esta solución logra una mejora de 26.25% y en el tiempo promedio requerido para evaluar cada una de las instancias logra una mejora de un 4.02%.

Capítulo 6

CONCLUSIONES Y TRABAJOS FUTUROS

En este trabajo se abordó el problema robusto del abastecimiento internacional con capacidad finita (ROCIS), el cual consiste en seleccionar un conjunto de proveedores para satisfacer la demanda de productos de un conjunto de plantas localizadas en diferentes países. También se presentan en este capitulo, las aportaciones obtenidas y las líneas de investigación identificadas en el desarrollo de este estudio.

6.1 Aportaciones 81
6.2 Trabajos futuros 82

6.1 Aportaciones

En el capítulo 4, se describen diversas estrategias de prioridades para la generación de soluciones iniciales de la solución tabú del problema ROCIS. Las soluciones reportadas en la literatura consideran dos estrategias de prioridades para elegir los proveedores que se incorporan a la solución inicial. La primera, consiste en dar prioridad a los proveedores de menor costo fijo y mayor capacidad de producción y la segunda, incorpora el valor esperado del costo de enviar productos del proveedor a todas las plantas. Una limitación de la primera alternativa, es que no se considera el costo de envío y aún cuando este factor sí es considerado por la segunda, el mecanismo que se utiliza resulta demasiado pesimista.

Otra aportación de este trabajo, es que, se documenta de manera detallada, la solución de referencia, con el propósito de que los resultados que se reportan, se puedan reproducir con relativa facilidad (ver Capítulo 3, Figura 3.1). Esta aportación es muy relevante, para que la investigación de la solución tabú del problema ROCIS, avance rápidamente como resultado de la participación de un mayor número de investigadores.

La principal aportación de este trabajo, es que se propone modificar el mecanismo de incorporación del costo de envío, para que se consideren únicamente las plantas hacia las que resulta más económico el envío de los productos desde el sitio del proveedor. Para validar este enfoque se proponen dos modelos alternativos de este concepto (ver Capítulo 4, secciones 4.2.3 y 4.2.4).

Los resultados experimentales muestran que la mejor estrategia propuesta logra reducir en un 3.32% el consumo de recursos requeridos para resolver las instancias y en un 21.05% los recursos requeridos para llegar a la mejor solución. Además de esta reducción en el consumo de recursos, se logra incrementar la calidad de la solución en un 1.14% (ver Capítulo 5, sección 5.2.4).

6.2 Trabajos futuros

En el desarrollo de esta investigación se han identificado las siguientes líneas de investigación para tratar de mejorar el desempeño de la solución de referencia:

Mejorar el mecanismo de generación de la vecindad y la eficiencia de la búsqueda local:

- Eficiencia de la vecindad

  Se propone redefinir el concepto de costo relativo incorporando información adicional de la instancia (posiblemente la capacidad de los proveedores) o información de los procesos de optimización con LINDO (posiblemente los costos reducidos).

- Eficiencia de la búsqueda local

  Se propone analizar de manera detallada el desempeño de los operadores actuales (inserción, eliminación e intercambio) para modificar su funcionalidad o incorporar nuevos operadores (reencadenamiento de trayectorias, genéticos, etc.).

A corto plazo se tiene proyectado construir un sitio Web dedicado exclusivamente a la investigación del problema ROCIS.

REFERENCIAS

[Anderson 1991] Anderson David R., Sweeney Dennos J., Williams Thomas A.: Introducción a los modelos cuantitativos para administración. Íberoamérica, México (1991), Capítulo 7

[Cohen 1989] Cohen, M.A. and H.L. Lee: Resource Deployment Analysis of Global Manufacturing and Distribution Networks, Journal of Manufacturing and Operations Management. Vol. 2. (1989) 81-104.

[Escudero 1993] Escudero, L.F., P.V. Kamesam, A.J. King y R.J.-B. Wets: Production Planning via Scenario Modelling, Annals of Operations Research. Vol. 43. (1993). 311-335.

[Glover 1997] Glover F, Laguna M. Tabu Search. Dordrecht: Kluwer Academic Publishers, 1997.

[González 2004] González-Velarde JL, Laguna M. A: Benders-based heuristic for the robust capacitated international sourcing problem. IIE Transactions. Vol 36. (2004). 1125-1133.

[González 2006] González-Velarde JL, Rafael Martí: Adaptive Memory Programming for the Robust Capacitated International Sourcing Problem.

[Gutiérrez 1995] Gutiérrez, G.J. and P. Kouvelis: A Robustness Approach to International Sourcing, Annals of Operations Research. Vol. 59. (1995). 165-193.

[Hodder 1982]. Hodder, J.E. and J.V. Jucker: Plant Location Modeling for the Multinational Firm, Proceedings of the Academy of International Business Conference on the Asia-Pacific Dimension of International Business. Honolulu. December. (1982). 248-258.

[Hodder 1986]. Hodder, J.E. and M.C. Dincer: A Multifactor Model for International Plant Location and Financing under Uncertainty, Computers & Operations Research. Vol. 13, no. 5. (1986). 601-609.

[Jucker 1976] Jucker, J.V. and R.C. Carlson: The Simple Plant-Location Problem under Uncertainty, Operations Research. Vol. 24, no. 6. (1976). 1045-1055.

[Kouvelis 1997] Kouvelis, P. y G. Yu: Robust Discrete Optimization and its Applications, Kluwer Academic Publishers, Dordrecht. (1997).

[Krarup 1979] Krarup, J. and P.M. Pruzan: Selected Families of Location Problems, Annals of Discrete Mathematics. Vol. 5. (1979). 327-387

[Lawrence 1999] Lawrence, S. R. and A. H. Buss: Excess Capacity and International Production Arbitrage, to appear in Advanced Manufacturing Systems: Strategic Management and Implementation, J. Sarkis. (1999).

[LINDO] LINDO Systems, Inc. LINDO API User`s Manual, Versión 2.0

[Louveaux 1992] Louveaux, F. V. and D. Peeters: A Dual-based Procedure for Stochastic Facility Location, Operations Research. Vol. 40, no. 3, (1992). 564-573.

[Martí 2006] Paquete de Instancias ROCIS , http://www.uv.es/~rmarti/paper/results/ROCIS%20Instances.zip

[Verter 1992] Verter, V. and M. C. Dincer: An Integrated Evaluation of Facility Location, Capacity Acquisition and Technology Selection for Designing Global Manufacturing Strategies," European Journal of Operational Research.Vol. 60. (1992).1-18.

SUMMARY

The robust capacitated international sourcing problem with a finite capacity is about selecting a set of suppliers to satisfy a demand of products in a set of plants which are located in different countries.

This paper analyzes different priority strategies in order to generate initial solutions applying the TABU search algorithm to the ROCIS problem. The aforementioned solutions are used to choose the suppliers to incorporate to an initial solution.

The first strategy used establishes a priority to the suppliers which have a lesser fixed cost and higher production capacity, whereas the second strategy incorporates the expected value of the cost of sending the products from the provider to all the plants.

The first strategy imposes a disadvantage due to the fact that the shipping cost is not considered and even if the second strategy does consider it, the mechanism used turns out to be very pessimistic.

This document suggests modifying the mechanism that incorporates the shipping cost so that the only plants considered are those whose shipping cost is cheaper. In order to validate the previous condition, two alternative models are proposed.

The experimental results show that one of the strategies that were tested renders a 3.32% reduction of consumed resources when solving the instances, and a 21.05% of the required resources in order to reach the best solution. In addition to the previously mentioned reduction, solution quality is increased by 1.14%.

Since the results are most promising, research continues currently in order to apply these strategies in the improvement of development of the ROCIS solution using path relinking.