Instituto Tecnológico de Cd. Madero

DIVISIÓN DE ESTUDIOS DE POSGRADO E INVESTIGACIÓN

TESIS

"Búsqueda Local Mejorada Para la Solución del 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. Miguel Colín Rodríguez

DIRECTORES DE TESIS:

Dr. Hector Joaquìn Fraire Huacuja

M.C. Guadalupe Castilla Valdéz

Opción I: Tesis Profesional

Ciudad Madero, Tamaulipas **Diciembre 2008**

Sistema Nacional de Educación Superior Tecnológica
Dirección General de Educación Superior Tecnológica

SUBSECRETARÍA DE EDUCACIÓN SUPERIOR
DIRECCIÓN GENERAL DE EDUCACIÓN SUPERIOR TECNOLÓGICA
INSTITUTO TECNOLÓGICO DE CIUDAD MADERO

SECRETARÍA DE EDUCACIÓN PUBLICA
SEP

Cd. Madero, Tam., a 18 de Noviembre de 2008.

Área: Posgrado
Nº Oficio: U5.394/08
Asunto: Autorización de Impresión de Tesis

C. ING. MIGUEL COLIN RODRÍGUEZ.
P r e s e n t e.

Me es grato comunicarle que después de la revisión realizada por el Jurado designado para su examen de grado de Maestro en Ciencias en Ciencias de la Computación, se acordó autorizar la impresión de su tesis titulada:

"BÚSQUEDA LOCAL MEJORADA PARA LA SOLUCIÓN DEL PROBLEMA ROBUSTO DE ABASTECIMIENTO INTERNACIONAL CON CAPACIDAD FINITA (ROCIS)"

Es muy satisfactorio para la División de Estudios de Posgrado e Investigación compartir con Usted el logro de esta meta. Espero que continúe con éxito su desarrollo profesional y dedique su experiencia e inteligencia en beneficio de México.

A t e n t a m e n t e
"POR MI PATRIA Y POR MI BIEN"

M.P. María Yolanda Chávez Cinco
Jefa de la División

S.E.P.
DIVISION DE ESTUDIOS DE POSGRADO E INVESTIGACION
I T C M

" 2008, Año de la Educación Física y el Deporte "

MYCHC ' NLCO ' a/m*

Ave. 1o. De Mayo y Sor Juana I. de la Cruz, Col. Los Mangos, C.P. 89440 Cd. Madero, Tam.
Teléfono : 01 (833) 357 48 20 al 29
Internet : www.itcm.edu.mx Correo Electrónico : itcm@itcm.edu.mx

TABLA DE CONTENIDO

LISTA DE TABLAS

LISTA DE FIGURAS

Capítulo 1

INTRODUCCIÓN

En este capítulo se presenta una visión general de esta tesis iniciando con una descripción de los motivos que justifican a esta investigación así como la definición del problema ROCIS. De igual manera se describen los objetivos, alcances y limitaciones.

1.1 Justificación

Debido al creciente comercio internacional y a las empresas transnacionales cuyas actividades tienen una interacción directa con otras similares en operaciones de compra-venta, se hace cada vez más compleja la elección de los proveedores que suministran la materia prima para generar su producto ya que los proveedores se encuentran localizados en diferentes países.

Los costos de envío varían uno de otro, la tasa de cambio de la moneda de los países es variante, el costo del producto de cada proveedor es diferente; la pregunta que se haría una persona encargada de comprar producto a un proveedor u otro sería, ¿a qué proveedor o proveedores es conveniente elegir y qué cantidad de producto comprar a cada uno de ellos con el fin de satisfacer la demandad requerida y a la vez minimizando los costos?

Como se puede apreciar, entre mayor sea el número de proveedores, el número de combinaciones posibles crece de manera exponencial haciendo esta labor de elección de proveedores un trabajo arduo y que no siempre se logra la solución óptima a dicho problema.

Es aquí donde dar solución al problema de abastecimiento internacional robusto con capacidad finita (ROCIS) tiene gran importancia, ya que reduciría el tiempo que se requiere para lograr la elección de proveedores y que ésta sea de manera óptima minimizando los costos asociados a dicha actividad.

Esta tesis aborda este problema con la finalidad de aportar soluciones que mejoren significativamente la solución al problema ROCIS, optimizando la elección de los proveedores, reduciendo el tiempo requerido y resolviendo de manera óptima las instancias de prueba descritas en [González 2004].

Esta investigación se encuentra en el contexto de búsqueda tabú [Glover 1986], que es un metaheurístico para dar solución a problemas de optimización como es el caso de este trabajo. Consiste de manera general en guiar un proceso de búsqueda local para explorar el espacio de soluciones más allá del óptimo local.

1.2 Descripción del problema

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. Con fines de simplificación para la modelación se considera un sólo producto en un sólo periodo. La incertidumbre de la demanda y la tasa de cambio se modelan utilizando un conjunto de escenarios. En el modelo se utilizan las siguientes definiciones:

Parámetros

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

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

S: Conjunto de escenarios.

f_i: Costos fijos asociados con el proveedor i.

c_{ij}: Costo unitario total de proporcionar artículos del proveedor i a la planta j.

b_i: Capacidad del proveedor i.

d_{js}: Demanda de la planta j en el escenario s.

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

p_s: Probabilidad de ocurrencia del escenario s.

Variables

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

y_i: Toma el valor de 1 si el proveedor i es contratado y 0 en caso contrario.

1.2.1 Subproblema de transporte asociado a cada escenario

Para cada elección de proveedores $y=[y_i]$ donde $i=1,2,....M$ se debe minimizar la función objetivo indicada en la ecuación (1.1).

$$\text{minimizar } z_s = \sum_{i \in M} \sum_{j \in N} e_{is} c_{ij} x_{ijs} \qquad (1.1)$$

en donde z_s representa el costo de transporte asociado a cada escenarios *s*. Dicha función debe estará sujeta a las restricciones:

$$\sum_{i \in M} x_{ijs} \geq d_{js} \quad \forall j \in N \qquad (1.2)$$

$$\sum_{i \in N} x_{ijs} \leq b_i y_i \quad \forall i \in M \qquad (1.3)$$

$$x_{ijs} \geq 0 \qquad \forall i \in M, j \in N \qquad (1.4)$$

en donde la restricciones (1.2) corresponde a la demanda de las plantas (d_{js}), la cual debe ser satisfecha; la restricción (1.3) corresponde a la capacidad del proveedor *i* (b_i), dicha capacidad no debe ser excedida; la restricción (1.4) corresponde a la cantidad de producto enviado al proveedor *i* por la planta *j* en el escenario *s*, dicha cantidad debe ser mayor que cero.

1.2.2 Función objetivo del problema ROCIS

En (1.5) se muestra la función objetivo del problema ROCIS.

$$\min \; F(y) = \sum_{s \in S} p_s \left(\sum_{i \in M} e_{is} f_i y_i + z_s \right) + \omega \sqrt{\frac{\sum_{s \in S^+} p_s \; z_s - E \; z \;^2}{\sum_{s \in S^+} p_s}} \qquad (1.5)$$

donde:

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

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

en donde $F \;\; y$ es la probabilidad de elegir al proveedor (y_i) con su costo fijo asociado (f_i) y el tipo de cambio de la localización del proveedor *i* en el escenario *s* (e_{is}), a esto se le suma el

costo de transporte asociado a cada escenario (z_s). Lo descrito anteriormente corresponde al primer termino de la función objetivo. El segundo termino penaliza solamente las desviaciones positivas del valor esperado ($E(z)$), en donde el valor esperado $E(z)$ es la sumatoria del producto de la probabilidad del ocurrencia del escenario *s* (p_s) y el costo de transporte asociado al escenario *s* (z_s).

Esto se lleva a cabo para aquellos escenarios en los cuales la diferencia entre el costo de transporte z_s y el valor esperado $E(z)$ es mayor o igual que cero. Dichos escenarios se encuentran contenidos en el conjunto S^+.

El valor de ω en el segundo termino de la ecuación es un factor que la persona encargada de tomar decisiones puede ajustar para dar mayor o menor importancia al componentes de riesgo de la función objetivo [González 2004].

1.3 Objetivo general

Mejorar la solución del problema de ROCIS en el contexto de búsqueda local.

1.3.1 Objetivos específicos

- Analizar la solución Tabú.
- Analizar la solución de Re-encadenamiento de trayectorias.
- Definir e implementar una estrategia de mejora.
- Evaluación experimental de la solución mejorada.

1.4 Alcances y limitaciones

La limitación correspondiente a la solución del problema consiste en que las implementaciones deben ser realizadas con base en la plataforma reportada en [Gómez 2007], la cuál es la siguiente: una computadora Dell Optiplex 160l con procesador Pentium

4 a 2.4 GHZ y 1 GB de memoria RAM. El compilador utilizado será Visual C 6.0 y el sistema operativo Windows XP. Para la solución de los subproblemas de transporte se utiliza LINDO API versión 2.0.

Los programas deben ser realizados en C estándar y soportar pruebas en Windows y Linux. Las instancias utilizadas para validar los resultados deben ser las mismas que se reportan en [González 2004].

1.5 Organización del documento

En el capítulo 2 se presenta el marco teórico que da fundamento a los diferentes elementos que conforman esta investigación. En primer lugar se presentan los conceptos de programación lineal, de manera específica el caso de transporte.

Posteriormente se definen conceptos básicos tales como el de matriz dispersa y la forma en que se da solución a los modelos de programación lineal con LINDO API. Se describe la optimización robusta y el re-encadenamiento de trayectorias. Finalmente se muestran los trabajos relacionados.

En el capítulo 3 se describe la propuesta de solución de esta tesis, así como una visión de lo que es búsqueda tabú y la solución tabú reportada en [González 2004].

El capítulo 4 muestra los resultados obtenidos durante la experimentación realizada, el análisis de dichos resultados y las gráficas comparativas del desempeño de la solución propuesta.

En el capítulo 5 se describen las aportaciones más importantes de esta tesis, las conclusiones y las líneas de investigación identificadas durante el proceso de investigación.

Capítulo 2

MARCO TEÓRICO

Este capítulo describe los fundamentos teóricos que dan sustento al trabajo realizado durante esta investigación. El tema de programación lineal es descrito, así como ejemplos para su fácil comprensión, posteriormente se aborda el tema de matrices dispersas, el uso de LINDO API para resolver los problemas de programación lineal y la manera en que se codifica en lenguaje C, de igual manera se describe en qué consiste la optimización robusta y por último se hace mención de los trabajos relacionados más relevantes, los cuales tratan algunas temáticas del problema ROCIS.

2.1 Programación lineal (PL)

La programación lineal consiste en optimizar (minimizar o maximizar) una función lineal, que se denomina función objetivo, las variables de dicha función pueden o no estar sujetas a una serie de restricciones que se expresan mediante un sistema de inecuaciones lineales. El problema de la resolución de un sistema lineal de inecuaciones se remonta, al menos, a Fourier, quien desarrollo el método de eliminación de Fourier-Motzkin. La programación lineal es un modelo matemático desarrollado durante la segunda guerra mundial para planificar los gastos y los retornos, a fin de reducir los costos al ejército y aumentar las pérdidas del enemigo. Se mantuvo en secreto hasta 1947. En la posguerra, muchas industrias lo usaron en su planificación diaria [Ignizio 1994].

Aunque basado en ideas, nociones y trabajos de numerosas investigaciones anteriores, la primera aproximación verdaderamente comprensiva y efectiva tanto para el modelado como para la solución de problemas de programación lineal fue desarrollada por George Dantzig y sus colegas en 1947 [Loomba 1964].

La programación lineal constituye un importante campo de la optimización por varias razones, muchos problemas prácticos de la investigación de operaciones pueden plantearse como problemas de programación lineal. Algunos casos especiales de programación lineal, tales como los problemas de flujo de redes y problemas de flujo de mercancías se consideraron en el desarrollo de las matemáticas lo suficientemente importantes como para generar por si mismos mucha investigación sobre algoritmos especializados en su solución. Una serie de algoritmos diseñados para resolver otros tipos de problemas de optimización constituyen casos particulares de la más amplia técnica de la programación lineal. Históricamente, las ideas de programación lineal han inspirado muchos de los conceptos centrales de la teoría de optimización tales como la dualidad, la descomposición y la importancia de la convexidad y sus generalizaciones. Del mismo modo, la programación lineal es muy usada en la microeconomía y la administración de empresas, ya sea para aumentar al máximo los ingresos o reducir al mínimo los costos de un sistema de producción.

En (2.1) se muestra la definición formal de un problema de programación lineal.

$$
\begin{aligned}
&\textit{Maximizar / Minimizar} \quad Z = \sum_{j=1}^{n} c_j x_j \\
&\text{Sujeto a:} \\
&\sum_{j=1}^{n} a_{ij} x_j \quad \{\leq, =, \geq\} \quad b_i \qquad i = 1, \ldots, m \\
&x_j \geq 0 \qquad j = 1, \ldots, n
\end{aligned}
\tag{2.1}
$$

Donde:

n número de variables de decisión

m número de restricciones

a Coeficiente de cada variable en las restricciones

b Valor ubicado en el lado derecho de las restricciones

c Coeficiente que denota la relación costo/beneficio de la variable de decisión.

x Variable de decisión

2.1.1 Área de aplicación de la programación lineal

Algunas de las áreas de aplicación de la programación lineal para dar solución a problemas son las siguientes:

- *Mezcla*. Por ejemplo, la mezcla de productos del petróleo para producir diferentes grados de gasolina, la mezcla de ingredientes para formar un producto químico en particular.
- *Planes de producción*. Por ejemplo, determinar cuántos artículos a producir cada periodo de tiempo así como para satisfacer la demanda de cliente, capacidad de producción, y limitaciones de almacenaje.
- *Asignación*. Por ejemplo, la asignación de trabajadores a tareas así como minimizar los costos o el tiempo.
- *Transporte / envío.* Por ejemplo, determinar el programa de envío necesario para

satisfacer la demanda del cliente mientras se minimiza los costos de transporte y establece un plan para recolección de desechos sólidos.

Dentro del problema de transporte se tienen algunos casos en particular:

- El problema de ruteo de vehículos
- El problema del agente viajero
- El problema del cartero

2.1.2 Caso especial de programación lineal: problema de transporte

En muchas aplicaciones, es necesario determinar un plan de envío para distribuir bienes desde una gran cantidad de almacenes (o centros de producción) a muchos distribuidores al por menor (o clientes). Debido a la proximidad y modo de transporte, el costo de enviar una unidad entre cada almacén y distribuidor puede variar de locación a locación. Además, los proveedores disponibles para enviar desde los almacenes y las unidades demandadas por los distribuidores al por menor también puede variar. La tarea entonces es determinar el número de unidades a enviar desde cada almacén para cada distribuidor mientras se minimiza el costo total de envío.

2.1.2.1 Un ejemplo del problema de transporte

Un fabricante tiene 3 almacenes que proveen del producto final a 4 distribuidores minoritarios. Los almacenes tienen 6000, 9000 y 4000 unidades disponibles respectivamente, y las demandas de los distribuidores están proyectadas que sean de 3900, 5200, 2700 y 6400 unidades respectivamente. Los costos de envío (en dólares) por unidad desde cada almacén hacia cada distribuidor están dados en la Tabla 2.1. El fabricante necesita determinar el plan de envío de costo mínimo que satisfaga toda la demanda.

Tabla 2.1. Costos de envío por unidad

	Distribuidores			
Almacén	1	2	3	4
1	7	3	8	4
2	9	5	6	3
3	4	8	9	6

2.1.2.1.1 Solución al ejemplo del problema de transporte

Determinar las variables de decisión. Las variables de decisión son las cantidades enviadas desde cada almacén a cada distribuidor:

x_{ij} = cantidad enviada desde el almacén *i* al distribuidor *j*: *i* = 1,2,3; *j* = 1,2,3,4.

Formulación de la función objetivo. La función objetivo es simplemente la suma de todas las unidades enviadas entre cada almacén y distribuidor multiplicada por los costos de envió por unidad. Por lo tanto, la función objetivo puede ser escrita como se muestra en (2.2):

$$\begin{aligned}\text{minimizar } z = {} & 7x_{11} + 3x_{12} + 8x_{13} + 4x_{14} + 9x_{21} + 5x_{22} \\ & + 6x_{23} + 3x_{24} + 4x_{31} + 6x_{32} + 9x_{33} + 6x_{34}\end{aligned} \qquad (2.2)$$

Formulación de las restricciones. Note que el número total de unidades que son enviadas desde el almacén *i* es dado por $x_{i1} + x_{i2} + x_{i3} + x_{i4}$. De igual manera, el número total de unidades que son enviadas al distribuidor *j* es $x_{1j} + x_{2j} + x_{3j}$. Así, debido a que los proveedores que están disponibles en cada almacén tienen limitada su capacidad, las restricciones de capacidad de los proveedores estarán dadas por:

$$x_{11} + x_{12} + x_{13} + x_{14} \leq 6000 \qquad (2.3)$$
$$x_{21} + x_{22} + x_{23} + x_{24} \leq 9000 \qquad (2.4)$$
$$x_{31} + x_{32} + x_{33} + x_{34} \leq 4000 \qquad (2.5)$$

Las restricciones indicadas en (2.3), (2.4 y (2.5) corresponden a las capacidades dadas de los almacenes 1, 2 y 3 respectivamente.

Las restricciones de la demanda para los distribuidores sigue una forma similar:

$$x_{11} + x_{21} + x_{31} = 3900 \quad (2.6)$$
$$x_{12} + x_{22} + x_{32} = 5200 \quad (2.7)$$
$$x_{13} + x_{23} + x_{33} = 2700 \quad (2.8)$$
$$x_{14} + x_{24} + x_{34} = 6400 \quad (2.9)$$

En este caso las restricciones correspondientes a las distribuidoras 1, 2,3 y 4 están dadas por las expresiones (2.6), (2.7), (2.8) y (2.9).

Note que las restricciones de no negatividad son:

$$x_{11}, x_{12}, x_{13}, x_{14}, x_{21}, x_{22}, x_{23}, x_{24}, x_{31}, x_{32}, x_{33}, x_{34} \geq 0 \quad (2.10)$$

El modelo completo por lo tanto, puede escribirse como se muestra en la Figura 2.1:

Minimizar $z = 7x_{11} + 3x_{12} + 8x_{13} + 4x_{14} + 9x_{21} + 5x_{22} + 6x_{23} + 3x_{24} + 4x_{31} + 6x_{32} + 9x_{33} + 6x_{34}$

Sujeto a

$$x_{11} + x_{12} + x_{13} + x_{14} \leq 6000$$
$$x_{21} + x_{22} + x_{23} + x_{24} \leq 9000$$
$$x_{31} + x_{32} + x_{33} + x_{34} \leq 4000$$
$$x_{11} + x_{21} + x_{31} = 3900$$
$$x_{12} + x_{22} + x_{32} = 5200$$
$$x_{13} + x_{23} + x_{33} = 2700$$
$$x_{14} + x_{24} + x_{34} = 6400$$
$$x_{11}, x_{12}, x_{13}, x_{14}, x_{21}, x_{22}, x_{23}, x_{24}, x_{31}, x_{32}, x_{33}, x_{34} \geq 0$$

Figura 2.1. Modelo de programación lineal.

Observe que todos los coeficientes de las variables en las restricciones (2.3 a 2.9) son

uno ó cero.

2.1.2.2 Solución mediante software

Para resolver el problema descrito anteriormente se utiliza el software LINDO API 2.0 (Linear Integer Discrete Optimizer). En la Figura 2.1 se muestra el script requerido por LINDO API como entrada. En primer lugar se define la función objetivo mostrada en (2.2), posteriormente se definen las restricciones de capacidad (2.3 a 2.5), y por último las restricciones de demanda (2.6-2.9). Observe que después de la función objetivo se introduce la etiqueta *S.T.*, esto indica que la función objetivo está sujeta a las restricciones que se definen posteriormente. Para indicar el final del srcipt se inserta la etiqueta *END*. Este script corresponde al modelo mostrado en la Figura(2.1).

```
min 7x11 + 3x12 + 8x13 + 4x14 + 9x21 + 5x22
+ 6x23 + 3x24 + 4x31 + 6x32 + 9x33 + 6x34
S.T.
x11+x12+x13+x14 <= 6000
x21+x22+x23+x24 <= 9000
x31+x32+x33+x34 <= 4000
x11+x21+x31 = 3900
x12+x22+x32 = 5200
x13+x23+x33 = 2700
x14+x24+x34 = 6400
END
```

Figura 2.2. Script requerido por LINDO API.

Una vez que se ha ejecutado en LINDO API el script mostrado en la Figura 2.2, se obtienen los resultados de salida que se muestran la Figura 2.3.

Como se puede apreciar, el valor óptimo para la función objetivo es de $66700. De esta manera se minimizan los costos de envío de los almacenes a los distribuidores. La solución dada por LINDO API se interpreta como se muestra en la Tabla 2.2.

```
 LP OPTIMUM FOUND AT STEP      5
 OBJECTIVE FUNCTION VALUE
 1)      66700.00

VARIABLE        VALUE       REDUCED COST
X11          0.000000          3.000000
X12       5200.000000          0.000000
X13          0.000000          1.000000
X14        100.000000          0.000000
X21          0.000000          6.000000
X22          0.000000          3.000000
X23       2700.000000          0.000000
X24       6300.000000          0.000000
X31       3900.000000          0.000000
X32          0.000000          3.000000
X33          0.000000          2.000000
X34          0.000000          2.000000
```

Figura 2.3. Resultados obtenidos por LINDO API.

Tabla 2.2. Cantidad de producto enviada del almacén *i* al distribuidor *j*.

	Distribuidores (*j*)			
Almacén (*i*)	1	2	3	4
1	0	5200	0	100
2	0	0	2700	6300
3	3900	0	0	0

El almacén 1 envía 5200 unidades al distribuidor 2 y 100 unidades al distribuidor 4. El almacén 2 envía 2700 unidades al distribuidor 3 y 6300 unidades al distribuidor 4. Finalmente el almacén 3 envía 3900 unidades al distribuidor 1, de esta manera se cumple la demanda de los distribuidores expresada en las restricciones (2.6 a 2.9).

2.1.2.3 Factores asociados a la programación lineal

Al resolver un problema de programación lineal, además de obtener los valores para las variables de decisión así como el valor de la función objetivo, se obtiene información adicional que puede ser de gran ayuda para conocer y analizar el problema en cuestión. Estos

elementos son: costos reducidos, variables de holgura, precios sombra, análisis de sensibilidad de los coeficientes y análisis de sensibilidad de las restricciones.

2.1.2.3.1 Costos reducidos (Reduced Cost)

Las variables que al obtener la solución dada por LINDO API tienen un valor diferente de cero, en este caso las variables $x_{12}, x_{14}, x_{23}, x_{24}$ y x_{31}, tienen un costo reducido de cero como se muestra en la Figura 2.4.

VARIABLE	VALUE	REDUCED COST
X12	5200.000000	0.000000
X14	100.000000	0.000000
X23	2700.000000	0.000000
X24	6300.000000	0.000000
X31	3900.000000	0.000000

Figura 2.4. Variables con costo reducido igual a cero.

En cambio las variables $x_{11}, x_{13}, x_{21}, x_{22}, x_{32}, x_{32}$ y x_{34} cuyo valor es cero tienen un costo reducido diferente de cero como lo muestra la Figura 2.5.

VARIABLE	VALUE	REDUCED COST
X11	0.000000	3.000000
X13	0.000000	1.000000
X21	0.000000	6.000000
X22	0.000000	3.000000
X32	0.000000	3.000000
X33	0.000000	2.000000
X34	0.000000	2.000000

Figura 2.5. Variables con costo reducido diferente de cero.

El significado del costo reducido para estas variables es la penalidad de agregar la variable en cuestión como parte de la solución, esto implica pagar el valor dado por el costo reducido por cada unidad de la variable a tratar. Si el sentido de la función objetivo es minimizar, la penalidad consiste en incrementar el costo de la función objetivo. Si el sentido de la función objetivo es maximizar, entonces la penalización consiste en disminuir el costo de la función objetivo.

Como ejemplo, consideremos la variable x_{22} de nuestro ejemplo de programación lineal, la Figura 2.6 muestra su costo reducido.

VARIABLE	VALUE	REDUCED COST
X22	0.000000	3.000000

Figura 2.6. Costo reducido para la variable x_{22} .

En este caso, la penalización consistiría en aumentar el costo de la función objetivo en $3 por cada unidad de la variable x_{22} . El valor óptimo de la función objetivo es de $66700 como se observa en la Figura 2.2, pero si se agrega la variable x_{22} como parte de la solución el valor de la función objetivo aumentaría a $66703, empeorando el resultado si tenemos en cuenta que el problema de ejemplo consiste en minimizar los costos.

2.1.2.3.2 Variables de holgura (Slacks o Surplus)

Las variables de holgura (slacks o surplus) indican en qué medida la solución óptima satisface las restricciones del modelo planteado para un problema dado.

En el caso de las restricciones que son menores o iguales ($\leq$) a las variables de holgura se les refiere como *Slack*. Cuando las restricciones son mayores o iguales ($\geq$), las variables de holgura son referenciadas como *Surplus*. En caso de que la restricción sea satisfecha de manera exacta para la igualdad, el valor de la variable de holgura ya sea *slack* ó *surplus* es cero. La Figura 2.7 muestra las variables de holgura.

ROW	SLACK OR SURPLUS
2)	700.000000
3)	0.000000
4)	100.000000
5)	0.000000
6)	0.000000
7)	0.000000
8)	0.000000

Figura 2.7. Variables de holgura.

Para ejemplificar lo anterior, la primera restricción de capacidad del almacén

mencionada anteriormente en (2.3), y en la solución dada por LINDO API como *ROW 2* (Figura 2.8), vemos que su variable de holgura que en este caso es un *slack* ya que la restricción es menor o igual que ($\leq$) tiene un valor de 700.

```
ROW    SLACK OR SURPLUS
2)           700.000000
```

Figura 2.8. Variables de holgura para la 1ª restricción de capacidad.

Esto indica que existen 700 unidades disponibles que podría enviar el almacén 1 a cualquier distribuidor que lo solicite.

$$x_{11} + \quad x_{12} + x_{13} + x_{14} \leq 6000 \qquad (2.11)$$

$$0 + 5200 + \quad 0 + 100 \leq 6000 \qquad (2.12)$$

Como se aprecia en (2.11) y (2.12) y de acuerdo al resultado dado por LINDO API en la Figura 2.2, se cumple la restricción de capacidad del almacén 1 al enviar solamente 5300 unidades de las 6000 disponibles, quedando un remanente de 700 unidades.

2.1.2.3.3 Precios sombra (Dual prices)

Los precios sombra representan el costo de producir o consumir un bien o servicio. En el caso del problema de ejemplo, el precio sombra representa la penalidad de pagar por unidad adicional de un recurso que se encuentra especificado en alguna de las restricciones. La Figura 2.9 muestra los precios sombra obtenidos de la solución dada por LINDO API.

```
ROW    DUAL PRICES
2)         0.000000
3)         1.000000
4)         0.000000
5)        -4.000000
6)        -3.000000
7)        -7.000000
8)        -4.000000
```

Figura 2.9. Precios sombra.

Los precios sombra se ven reflejados de manea directa en el incremento o decremento

de la función objetivo. Dicho incremento o decremento depende del tipo de objetivo del problema (minimizar o maximizar) y del valor del precio sombra, ya sea este positivo o negativo.

Para entender el comportamiento de los precios sombra con respecto a la calidad de la solución, la Tabla 2.3 describe de que manera actúan dependiendo del tipo de objetivo y de su valor.

Tabla 2.3. Comportamiento de los precios sombra.

Tipo objetivo	**Precio Sombra**	
	Positivo	**Negativo**
Maximizar	Incrementa el valor de la función objetivo	Decrementa el valor de la función objetivo
Minimizar	Decrementa el valor de la función objetivo	Incrementa el valor de la función objetivo

Para ejemplificar lo descrito anteriormente, consideremos la restricción 2 referenciada en la solución dada por LINDO API como *ROW 3* (Figura 2.10), en este caso vemos que su precio sombra es de 1.000.

```
ROW    SLACK OR SURPLUS  DUAL PRICES
3)         0.000000        1.000000
```

Figura 2.10. Precios sombra para la 2ª restricción de capacidad.

El significado del precio sombra se puede interpretar de la siguiente manera: dado que ya no existen unidades disponibles del recurso debido a que su *slack* es cero, entonces por cada unidad de recurso adicional que se considere en la restricción (2.4) la función objetivo disminuirá en $1mejorando de esta manera la solución.

$$x_{21} + x_{22} + x_{23} + x_{24} \leq 9000$$
$$x_{21} + x_{22} + x_{23} + x_{24} \leq 9001 \qquad (2.13)$$

Como se aprecia en (2.13), si el lado derecho de la restricción aumenta en 1, el valor de la función objetivo disminuirá de \$66700 a \$66699 mejorando dicho valor.

2.1.2.3.4 Análisis de sensibilidad de los coeficientes de la función objetivo

El análisis de sensibilidad de los coeficientes de las variables de la función objetivo mostrado en la Figura 2.11, permite determinar en cuántas unidades se puede incrementar o decrementar el valor dichos coeficientes sin alterar el valor de la función objetivo.

RANGES IN WHICH THE BASIS IS UNCHANGED:

	OBJ COEFFICIENT RANGES		
VARIABLE	CURRENT COEF	ALLOWABLE INCREASE	ALLOWABLE DECREASE
X11	7.000000	INFINITY	3.000000
X12	3.000000	3.000000	INFINITY
X13	8.000000	INFINITY	1.000000
X14	4.000000	1.000000	1.000000
X21	9.000000	INFINITY	6.000000
X22	5.000000	INFINITY	3.000000
X23	6.000000	1.000000	INFINITY
X24	3.000000	1.000000	1.000000
X31	4.000000	3.000000	INFINITY
X32	6.000000	INFINITY	3.000000
X33	9.000000	INFINITY	2.000000
X34	6.000000	INFINITY	2.000000

Figura 2.11. Análisis de sensibilidad de los coeficientes de la función objetivo.

Como ejemplo, consideremos la variable x_{21} cuyo coeficiente en la función objetivo es 9, tal como se muestra en la expresión (2.14).

$$\begin{aligned} \text{minimizar } z = {} & 7x_{11} + 3x_{12} + 8x_{13} + 4x_{14} + 9x_{21} + 5x_{22} \\ & + 6x_{23} + 3x_{24} + 4x_{31} + 6x_{32} + 9x_{33} + 6x_{34} \end{aligned} \qquad (2.14)$$

El análisis de sensibilidad para dicha variable es descrito en la Figura 2.12 y

observamos que el incremento permitido para la variable x_{21} es *INFINITO* y su decremento permitido es de 6 sin que alguno de estos cambios modifique el valor óptimo de la función objetivo.

```
RANGES IN WHICH THE BASIS IS UNCHANGED:

                    OBJ COEFFICIENT RANGES
VARIABLE    CURRENT        ALLOWABLE        ALLOWABLE
             COEF          INCREASE         DECREASE
X21        9.000000        INFINITY         6.000000
```

Figura 2.12. Análisis de sensibilidad de los coeficientes de la función objetivo.

Por lo tanto, el rango en el que la variable x_{21} puede ser modificada sin que afecte al valor óptimo de la función objetivo es $6 \le x_{21} \le \infty$. Esto se puede comprobar cambiando el coeficiente de la variable x_{21} en la función objetivo mostrada en (2.14), dándole un valor de x_{21}=90, quedando la función objetivo como se aprecia en (2.15).

$$\begin{aligned} \text{minimizar } z = {} & 7x_{11} + 3x_{12} + 8x_{13} + 4x_{14} + 90x_{21} + 5x_{22} \\ & + 6x_{23} + 3x_{24} + 4x_{31} + 6x_{32} + 9x_{33} + 6x_{34} \end{aligned} \qquad (2.15)$$

Resolviendo con LINDO API se obtienen los resultados mostrados en la Figura 2.13.

```
LP OPTIMUM FOUND AT STEP      5
OBJECTIVE FUNCTION VALUE
1)      66700.00

VARIABLE        VALUE          REDUCED COST
X11           0.000000           3.000000
X12        5200.000000           0.000000
X13           0.000000           1.000000
X14         100.000000           0.000000
X21           0.000000           6.000000
X22           0.000000           3.000000
X23        2700.000000           0.000000
X24        6300.000000           0.000000
X31        3900.000000           0.000000
X32           0.000000           3.000000
X33           0.000000           2.000000
X34           0.000000           2.000000
```

Figura 2.13. Resultados obtenidos por LINDO API modificando el coeficiente de la variable x_{21}.

Si comparamos los resultados mostrados en la Figura 2.2 con los de la Figura 2.13, podemos apreciar que el valor óptimo de la función objetivo no cambia a pesar de ser modificado el coeficiente de la variable x_{21}, de 9 a 90.

2.1.2.3.5 Análisis de sensibilidad de los recursos disponibles (RHS)

Mediante el análisis de sensibilidad de los recursos disponibles de las restricciones (RHS – Right Hand Side) se puede determinar en cuantas unidades puede ser incrementado o decrementado el recurso de cada restricción sin alterar los valores de las variables de la solución óptima. La Figura 2.14 muestra dicho análisis de sensibilidad obtenido mediante LINDO API.

	RIGHTHAND SIDE RANGES		
ROW	CURRENT RHS	ALLOWABLE INCREASE	ALLOWABLE DECREASE
2	6000.000000	INFINITY	700.000000
3	9000.000000	100.000000	700.000000
4	4000.000000	INFINITY	100.000000
5	3900.000000	100.000000	3900.000000
6	5200.000000	700.000000	5200.000000
7	2700.000000	700.000000	100.000000
8	6400.000000	700.000000	100.000000

Figura 2.14. Análisis de sensibilidad de los recursos disponibles RHS.

Como ejemplo, consideremos la restricción de capacidad 1 mostrada en (2.16) y referenciada como *ROW 2* en la solución dada por LINDO API.

$$x_{11} + x_{12} + x_{13} + x_{14} \leq 6000 \qquad (2.16)$$

El análisis de sensibilidad para la restricción 1 es descrita en la Figura 2.15.

	RIGHTHAND SIDE RANGES		
ROW	CURRENT RHS	ALLOWABLE INCREASE	ALLOWABLE DECREASE
2	6000.000000	INFINITY	700.000000

Figura 2.15. Análisis de sensibilidad de los recursos disponibles RHS de la restricción 1.

Como se puede apreciar, el valor actual (Current RHS) de la restricción 1 es 6000 y su incremento permitido es de *INFINITO* y su decremento permitido es de 700 unidades. Por lo tanto, el rango en el que el valor de la restricción puede variar sin afectar el valor de las variables es $5300 \leq Capacidad\ de\ la\ restricción\ 1 \leq \infty$.

Esto puede ser comprobado modificando el valor actual de la restricción 1 de 6000 a 7000. La restricción 1 con el valor modificado se muestra en (2.17).

$$x_{11} + x_{12} + x_{13} + x_{14} \leq 7000 \qquad (2.17)$$

Resolviendo con LINDO API se obtienen los resultados mostrados en la Figura 2.16.

```
LP OPTIMUM FOUND AT STEP      0
OBJECTIVE FUNCTION VALUE
1)      66700.00

VARIABLE        VALUE          REDUCED COST
X11            0.000000          3.000000
X12         5200.000000          0.000000
X13            0.000000          1.000000
X14          100.000000          0.000000
X21            0.000000          6.000000
X22            0.000000          3.000000
X23         2700.000000          0.000000
X24         6300.000000          0.000000
X31         3900.000000          0.000000
X32            0.000000          3.000000
X33            0.000000          2.000000
X34            0.000000          2.000000
```

Figura 2.16. Resultados obtenidos por LINDO API modificando el valor de la restricción 1.

Si comparamos los resultados obtenidos por LINDO API y que se muestran en la Figura 2.2 con los obtenidos al hacer la modificación de la restricción 1 descritos en la Figura 2.16, podemos apreciar que el valor óptimo de la función objetivo no se alteró.

De igual manera, el valor de las variables de decisión permaneció sin cambio a pesar de haber modificado el valor de la restricción 1.

2.2 Matrices dispersas

Una matriz dispersa es una matriz que tiene un número relativamente pequeño de elementos

distintos a cero. Se representa la matriz usando tres (u opcionalmente cuatro) vectores. Este esquema se utiliza para reducir drásticamente los requerimientos de almacenaje. La representación de la matriz dispersa no almacena los coeficientes cero. Dado que la mayoría de los coeficientes de la matriz en modelos de programación matemática del mundo real son cero, este esquema del almacenaje resulta ser muy eficiente. A continuación se muestra una breve explicación del esquema de representación. Se utiliza la siguiente matriz de la Figura 2.17 para los ejemplos:

$$\begin{bmatrix} 3 & 0 & 0 & 2 \\ 0 & 6 & 0 & 9 \\ 4 & 5 & 8 & 0 \\ 0 & 7 & 1 & 0 \end{bmatrix}$$

Figura 2.17. Ejemplo de matriz dispersa

2.2.1 Representación de 3 vectores

Mediante tres vectores es posible representar una matriz dispersa. Un vector contendrá todas las entradas distintas a cero de la matriz, ordenadas por columna. En el ejemplo, este vector sería:

[3 4 6 5 7 8 1 2 9]

Figura 2.18. Vector Valor.

Observe que los elementos 3 y 4 correspondientes a la primera columna son los primeros en aparecer, sus índices son 0 y 1 respectivamente. Posteriormente aparecen los elementos de la segunda columna que son el 6, 5 y 7 que tendrán los índices 2, 3 y 4 respectivamente, este proceso continúa hasta vaciar todos los elementos de la matriz en el vector. Todos los elementos de la matriz con valor igual a 0 han sido descartados. El vector que contiene los datos de la matriz condensados se denomina vector Valor.

En el segundo vector, denominado Vector Inicio - Columna, se registran las entradas que en el vector Valor representan el comienzo de una columna nueva de la matriz original. La

Figura 2.19 muestra subrayados los valores que comienzan columna.

[3 4 6 5 7 8 1 2 9]

Figura 2.19. Vector Valor con elementos que inician columna subrayados.

Se utiliza la cuenta basada en cero, el vector Inicio - Columna es:

[0 2 5 7 9]

Figura 2.20. Vector Inicio – Columna.

En el vector Inicio – Columna el primer elemento nos indica que el elemento del vector Valor con índice 0 es el que inicia la primera columna de la matriz original. El segundo elemento del vector Inicio – Columna indica que el elemento con índice 2 en el vector Valor inicia la segunda columna. De igual manera, el tercer elemento del vector Inicio – Columna indica que el elemento con índice 5 del vector Valor inicia la tercera columna. El cuarto elemento del vector Inicio – Columna indica que el elemento del vector Valor con índice 7 inicia la cuarta columna de la Figura 2.17.

Observe que el vector Inicio - Columna mostrado en la Figura 2.20 tiene una entrada más que el número de columnas en nuestra matriz. La entrada adicional nos indica donde termina la última columna. Ésta última entrada será siempre igual a la longitud del vector Valor. Del vector Inicio - Columna, podemos deducir que entrada en el vector Valor pertenece a una determinada columna.

La única información adicional que se requiere es el número de fila para cada entrada. Almacenamos esta información en un tercer vector, el vector Índice - Fila. Este vector es de la misma longitud que el vector Valor. Cada entrada en el vector Índice - Fila indica a qué fila pertenece la entrada correspondiente del vector Valor. Por ejemplo, el elemento con valor de 3 en la matriz original (Figura 2.17) pertenece a la primera fila, que se denomina la fila 0, así que la primera entrada en el vector del Índice - Fila es 0. De la misma manera, la segunda entrada en el vector Valor (4), pertenece a la tercera fila (fila 2 al contar desde cero), así que la segunda entrada del vector Índice - Fila es 2, este proceso continúa, obteniéndose el vector Índice – Fila que se muestra en la Figura 2.21:

[0 2 1 2 3 2 3 0 1]

Figura 2.21. Vector Índice - Fila.

2.2.2 Ejemplo de la matriz de restricciones

La Figura 2.22 muestra un ejemplo de un problema de programación lineal.

$$
\begin{aligned}
& MAX = 20*A + 30*C \\
& S.T. \\
& \quad A + 2*C \leq 120 \\
& \quad A \leq 60 \\
& \quad C \leq 50
\end{aligned}
$$

Figura 2.22. Ejemplo de un problema de programación lineal.

la representación en 3 vectores de la matriz dispersa es la que se muestra a continuación:

- Vector valor: [1 1 2 1]
- Vector inicio-columna: [0 2 4]
- Vector índice-fila: [0 1 0 2]

2.3 Solución de modelos utilizando LINDO API 2.0

LINDO (Linear Integer Discrete Optimizer) es una herramienta para desarrolladores, la cual se puede incorporar a los programas de aplicaciones de optimización en diversos lenguajes de programación mediante la invocación de funciones propias. LINDO API está diseñado para resolver problemas de optimización, incluyendo programación lineal, programación entera, programación cuadrática y en general, programación no-lineal o no-convexa [LINDO API 2002].

2.3.1 Estructuras requeridas por LINDO API

Para dar solución a problemas de programación lineal, LINDO utiliza ciertas estructuras, las cuáles son enviadas a las funciones predeterminadas en dicha herramienta.

A continuación se muestran las estructuras antes mencionadas.

Tabla 2.4. Estructuras requeridas por LINDO API.

Dato	Tipo	Descripción
pachContypes	Char *	tipo de restricción (L – menor o igual que, E – igual que, G – mayor o igual que)
nCons	Int	numero de restricciones
padC	Double *	coeficientes de la función objetivo
nAnnz	Int	numero de no ceros en la matriz de restricciones
padAcoef	Double *	coeficientes de los no ceros en la matriz de restricción
paiArows	Int *	índice de fila de los no ceros
nVars	Int	numero de variables
paiAcols	Int *	índice del primer no ceros en cada columna
padB	Double *	coeficientes del lado derecho de las restricciones
padL	Double *	limites bajos de cada variable
padU	Double *	limites altos de cada variable
dObjconst	Double	valor constante agregado a la función objetivo
pacAcols	Int *	longitud de cada columna
tipo_var	Int *	tipo de variable, 1 si es entera, 0 si no lo es

2.3.2 Ejemplo de representación de un problema de programación lineal

Dado el problema de programación lineal entera que se muestra en la Figura 2.22, se muestra el contenido de las estructuras requeridas por LINDO API.

El contenido de cada una de las estructuras que requiere LINDO API para el problema en cuestión se muestra en la Tabla 2.5.

Tabla 2.5. Representación de un problema en las estructuras requeridas por LINDO API.

Dato	Descripción	Contenido
pachContypes	tipo de restricción (L – menor o igual que, E – igual que, G – mayor o igual que)	L, L, L
nCons	numero de restricciones	3
padC	coeficientes de la función objetivo	20, 30
nAnnz	numero de no ceros en la matriz de restricciones	4
padAcoef	coeficientes de los no ceros en la matriz de restricción	1, 1, 2, 1
paiArows	índice de fila de los no ceros	0, 1, 0, 2
nVars	numero de variables	2
paiAcols	índice del primer no ceros en cada columna	0, 2, 4
padB	coeficientes del lado derecho de las restricciones	120, 60, 50
padL	limites bajos de cada variable	Null
padU	limites altos de cada variable	Null
dObjconst	valor constante agregado a la función objetivo	0
pacAcols	longitud de cada columna	Null
tipo_var	tipo de variable, 1 si es entera, 0 si no lo es	1

2.3.3 Codificación en lenguaje C

Para dar solución a un problema de programación lineal con LINDO API utilizando el lenguaje de programación C, se deben seguir los pasos que se describen a continuación:

1. Crear un entorno de LINDO.

2. Crear un modelo en el entorno de LINDO.

3. Especificar o definir el modelo utilizando las estructuras descritas en la Tabla 2.4.

4. Ejecutar la optimización.

5. Recuperar los resultados del modelo a optimizar.

6. Eliminar el entorno de LINDO.

La Figura 2.23 muestra el código requerido para dar solución al ejemplo mostrado en la Figura 2.22.

```
#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("Errorcode=%d:  %s\n", nErrorCode, \
         cErrorMessage); \
      } else {\
         printf( "Fatal Error\n"); \
      } \
      exit(1); \
   } \
int main()
{  APIERRORSETUP;
/* Número de restricciones */
   int nM = 3;
/* Número de variables */
   int nN = 2;
```

Figura 2.23. Código fuente del problema de ejemplo.

```
/* Declarar una objeto para el entorno de LINDO*/
   pLSenv pEnv;
/* Declarar un objeto para el modelo de LINDO */
   pLSmodel pModel;
   int nSolStatus;
/* Paso 1. Crear un entorno de LINDO */
   pEnv = LScreateEnv ( &nErrorCode, MY_LICENSE_KEY);
   if ( nErrorCode == LSERR_NO_VALID_LICENSE)
   {  printf( "Invalid License Key!\n");
      exit( 1);
   }
   APIERRORCHECK;
/* Paso 2. Crear un modelo en el entorno. */
   pModel = LScreateModel ( pEnv, &nErrorCode);
   APIERRORCHECK;
   {
/* Paso 3. Especificar el modelo */
/* Dirección de optimización */
      int nDir = LS_MAX;
/* Término constante de la función objetivo */
      double dObjConst = 0.;

/* Coeficientes de la función objetivo */
      double adC[2] = { 20., 30.};
/* Lado derecho de las restricciones */
      double adB[3] = { 120., 60., 50.};
/* Tipo de restricciones */
      char acConTypes[3] = {'L', 'L', 'L'};
/* Número no cero en la matriz de restricciones */
      int nNZ = 4;
/* Vector Inicio - Columna de la matriz de restricciones */
      int anBegCol[3] = { 0, 2, nNZ};
/* Vector Valor de la matriz de restricciones */
      double adA[4] = { 1., 1., 2., 1.};
/* Vector Índice - Fila de la matriz de restricciones */
      int anRowX[4] = { 0, 1, 0, 2};
/* Longitud de cada columna (se debe especificar cuando la
   matriz de restricciones incluya elementos que son cero, en
   otro caso se le asigna NULL) */
      int *pnLenCol = NULL;
/* Límites inferiores y superiores en las variables. Los valores
   por defecto son cero o infinito. Para utilizar dichos valores
   se pasa NULL a los apuntadores */
      double *pdLower = NULL, *pdUpper = NULL;
/* Cargar el modelo a memoria con la función LSloadLPData */
      nErrorCode = LSloadLPData( pModel, nM, nN, nDir,
      dObjConst, adC, adB, acConTypes, nNZ, anBegCol,
       pnLenCol, adA, anRowX, pdLower, pdUpper);
      APIERRORCHECK; }
```

Figura 2.23. Continuación …

```
/* Paso 4. Ejecutar la optimización */
  nErrorCode = LSoptimize( pModel,
   LS_METHOD_PSIMPLEX, &nSolStatus);
  APIERRORCHECK;
  if (nSolStatus == LS_STATUS_OPTIMAL ||
      nSolStatus == LS_STATUS_BASIC_OPTIMAL){
/* Paso 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( "Objective Value = %g\n", dObj);
/* Obtener el valor de las variables de decisión*/
     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 encontrada -- status:
               %d\n", nSolStatus); }
/* Paso 6. Eliminar el entorno de LINDO */
  nErrorCode = LSdeleteEnv( &pEnv);
  printf("Presionar <Enter> ...");
  getchar(); }
```

Figura 2.23. Continuación…

2.4 Optimización robusta

Ha habido, en años recientes, un considerable interés en soluciones robustas para muchos problemas de decisión. Este creciente interés es motivado por el hecho de que algunos parámetros importantes de esos problemas de decisión dependen en gran manera de la cambiante comprensión del futuro. Recientemente, en [Mulvey 1995] se introdujo la noción de optimización robusta, un marco para direccionar el problema de encontrar soluciones robustas a algunos problemas estocásticos de optimización.

Usando la terminología estándar en programación estocástica. Dado que *x* denota las variables de diseño cuyos valores óptimos son independientes de cualquier comprensión de los parámetros inciertos, entonces dado *y* denota las variables de control las cuales pueden ser ajustadas una vez que los parámetros inciertos son determinados.

Los valores óptimos de las variables de control dependen tanto de la comprensión de los parámetros inciertos como de los valores de las variables de diseño. Ahora consideremos el siguiente problema de programación lineal (LP):

$$\begin{aligned} \text{(LP) : Minimize} \quad & cx + dy \\ \text{Subject to:} & \\ & Ax = b && (2.18) \\ & Ex + Fy = g && (2.19) \\ & x, y \geq 0 && (2.20) \end{aligned}$$

donde *c,d,A,b,E,F* y *g* son parámetros que definen las entradas del modelo. La restricción (2.18) denota las restricciones de diseño que no son impactadas por la incertidumbre. La restricción (2.19) modela las restricciones de control en las cuales los coeficientes pueden estar sujetos a incertidumbre.

Para definir un problema de optimización robusta asociado con un problema de programación matemática LP, sea $PS = \{1, 2, \ldots, S\}$ un conjunto de posibles escenarios futuros, cada uno de los cuáles contiene una probabilidad de ocurrencia p_s tal que $\sum_{s \in PS} p_s = 1$. También, para cada escenario $s \in PS$, sea $\{E_s, F_s, d_s, g_s\}$ el conjunto de realizaciones para los coeficientes de las restricciones de control y la función objetivo del programa matemático LP.

Una solución óptima del programa matemático LP es considerada una "solución robusta", si permanece cercana al óptimo para cualquier realización del escenario $s \in PS$. Si esta solución permanece "casi" factible para toda realización de $s \in PS$, este es considerado un "modelo robusto". Por supuesto, es inverosímil que una solución para el problema LP permanezca tanto factible como óptimo para cualquier realización de $s \in PS$ es necesario para un modelo robusto medir la compensación entre la solución y la robustez del modelo. El siguiente modelo formaliza una manera de medir esta compensación.

Dado $\{y_1, y_2, \ldots, y_s\}$ las variables de control para cada escenario $s \in PS$. Dada la realización de diferentes escenarios, no existe garantía de que (2.19) siempre sea satisfecha. Las "variables de error" $\{e_1, e_2, \ldots, e_s\}$ son introducidas para medir la infactibilidad en las

restricciones de control (2.19) bajo el escenario *s*. Ahora consideremos la siguiente reformulación del modelo de optimización robusta RP:

$(\mathrm{LP}_{\mathrm{RO}})$: Minimize $\sigma(x, y_1, y_2, \ldots, y_s) + \omega\rho(e_1, e_2, \ldots, e_s)$

Subject to:

$$Ax = b \tag{2.18}$$

$$E_s x + F_s y + e_s = g_s \tag{2.21}$$

$$x, y_s \geq 0 \tag{2.20}$$

El primer término de la función objetivo mide la robustez de la solución, y el segundo término mide la robustez del modelo. El término ω es un peso que mide la importancia relativa de obtener una solución enfocada a la robustez del modelo contra una solución enfocada a la robustez de la solución. Note que hay una variedad de elección para las funciones de costos $\sigma(x, y_1, y_2, \ldots, y_s)$ y la función de penalización de factibilidad $\rho(e_1, e_2, \ldots, e_s)$ usada para penalizar las violaciones de las restricciones de control. Sin embargo, es necesario que las funciones elegidas guíen a preferencias consistentes entre decisiones alternativas.

Un ejemplo de medida de la robustez de la solución es la función máxima de peso $\sigma(x, y) = \max_{s \in PS} \ \xi_s - \xi_s^*$ donde ξ es la función objetivo a ser minimizada. El valor ξ_s es el valor de la función objetivo, dado un determinado plan de diseño (por ejemplo, plan particular de expansión), entonces el escenario s toma un valor de verdadero. El valor ξ_s^* es el valor de la función objetivo para el plan óptimo para el cual sabemos que este escenario seria verdadero. Esta medida asegura que la mayor desviación esperada de la solución robusta de la solución óptima del escenario es minimazada. Alternativamente, una fuerza positiva de la diferencia entre el valor objetivo de implementar el plan de diseño y el objetivo del escenario óptimo podría ser usado:

$$\sigma(x, y) = \sum_{s \in PS} \ \xi_s - \xi_s^* \ ^q \tag{2.22}$$

Esta función se reduce al minimzar el peso esperado cuando q es igual a 1, y es igual al objetivo de programación lineal estocástica. Los valores de *q* mayores que 1 llevan a medidas de riesgo de la robustez de la solución.

2.5 Re-encadenamiento de trayectorias

La estrategia de re-encadenamiento de trayectorias (RT) o path relinking [González 2006], puede ser considerada como una extensión de los mecanismos clásicos de combinación de métodos evolutivos. En lugar de producir directamente una nueva solución cuando se combinan 2 o más soluciones originales, RT genera trayectorias entre y más allá de las soluciones seleccionadas en el espacio de la vecindad. El carácter de tales trayectorias se especifica mediante la referencia a los atributos de la solución que son agregados o de otra manera modificados por los movimientos ejecutados. Ejemplos de tales atributos incluyen aristas y nodos de un grafo, posiciones de secuencia en un horario, vectores contenidos en soluciones de programación lineal básica, y valores de variables y funciones de variables.

Para generar las trayectorias deseadas, es necesario seleccionar movimientos que desempeñen la siguiente función: comenzando con una *solución inicial*, los movimientos deben progresivamente introducir atributos que son proporcionados por una *solución guía* (o reducir la distancia entre atributos de las soluciones iniciales y guía). Entonces, considerar la creación de una trayectoria que una 2 soluciones seleccionadas y' y y'', produciendo una secuencia de solución $y' = y(1), y(2), \ldots, y(r) = y''$. RT inicia desde un conjunto dado de soluciones elite obtenidas durante un proceso de búsqueda. Para cada par de soluciones y' y y'', primero se calculan dos nuevas soluciones, $y_{\cap}$ y $y_{\cup}$. En la $y_{\cap}$ se selecciona los proveedores presentes en ambas soluciones y $y_{\cup}$ contiene aquellos proveedores presentes en al menos alguna de ellas. En términos matemáticos:

$$y_{\cap i} = \min \left\{ y'_i, y''_i \right\} \ , \ y_{\cup i} = \max \left\{ y'_i, y''_i \right\} \qquad (2.22)$$

Entonces, en vez de crear una trayectoria entre y' y y'', se crea un trayectoria entre estos 2 nuevos puntos $y_{\cap}$ y $y_{\cup}$. Dado $y_{\cap}$ sea la *solución inicial* y $y_{\cup}$ sea la *solución guía* en la trayectoria (la cual contiene a y' y y'' como se muestra en la Figura 2.24). Dado S' sea el conjunto de proveedores seleccionados en y' pero no seleccionados en y''. Simétricamente, dado S'' sea el conjunto de proveedores no seleccionados en y' y seleccionados en y''.

$$S' = \text{ proveedor } j \left| y'_j = 1 \text{ y } y''_j = 0 \right. \text{ , } S'' = \text{ proveedor } j \left| y'_j = 0 \text{ y } y''_j = 1 \right. \qquad (2.23)$$

Iniciando con $y_{\cap}$, las soluciones intermedias en la primera parte de la trayectoria son generadas agregando un proveedor de S' a la solución actual hasta alcanzar la solución y'. Una vez que y' ha sido alcanzada (después de $|S'|$ selecciones), se alterna entre agregar y eliminar proveedores de la solución actual para alcanzar y''. Específicamente, los proveedores que deben ser eliminados son aquellos que se encuentran en el conjunto S', mientras que los proveedores que son agregados son aquellos de S''. En las iteraciones pares se agregan proveedores y en iteraciones impares se eliminan.

Finalmente, una vez que y'' ha sido alcanzada, en la tercera parte de la trayectoria, se agregan proveedores en S' para obtener $y_{\cup}$. La Figura 2.24 muestra de manera grafica el re-encadenamiento de trayectorias entre 2 soluciones dadas y' y y''.

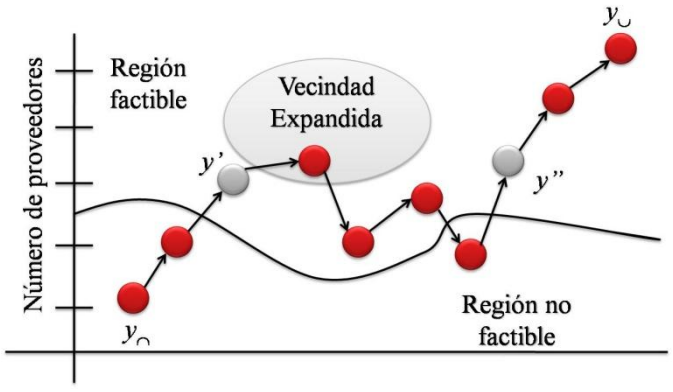

Figura 2.24. Ilustración de re-encadenamiento de trayectorias.

2.5.1 Algoritmo re-encadenamiento de trayectorias

De manera concisa se presenta a continuación el algoritmo del re-encadenamiento de trayectorias.

1. Para cada par de soluciones y' y y'' de proveedores.
    a. Calcular dos nuevas soluciones $y_{\cap}$ y $y_{\cup}$ donde $y_{\cap i} = 1$ sí $y'_i = 1$ y $y''_i = 1$, 0 en caso contrario, $y_{\cup i} = 1$ sí $y'_i = 1$ ó $y''_i = 1$, 0 en caso contrario.
    b. Calcular los conjuntos S' (proveedores seleccionados en y' pero no seleccionados en y'') y S'' (proveedores no seleccionados en y' y seleccionados en y'').
    c. Agregar a la solución $y_{\cap}$, los proveedores del conjunto S' de uno en uno en el orden dado hasta alcanzar a y'.
    d. Una vez alcanzada y' se alternará entre eliminar en las iteraciones impares a un proveedor de S' y agregar a un proveedor de S'' en las iteraciones pares hasta alcanzar a y''.
    e. Agregar a la solución y'', los proveedores del conjunto S' de uno en uno en el orden dado hasta alcanzar a $y_{\cup}$.
    f. Evaluar todas las soluciones resultantes y guardar aquellas que obtengan un mejor valor que y' y y''.
2. Devolver la solución con el mejor valor.
3. Fin

2.5.2 Ejemplo de re-encadenamiento de trayectorias

Dado el par de soluciones iniciales

$$y' = [\,0\,1\,1\,0\,0\,1\,0\,1\,0\,1\,], \; y'' = [\,1\,0\,1\,0\,1\,1\,0\,0\,1\,1\,]$$

de estas se obtienen 2 nuevas soluciones, $y_{\cap}$ y $y_{\cup}$:

$$y_{\cap} = [\,0\,0\,1\,0\,0\,1\,0\,0\,0\,1\,],\quad y_{\cup} = [\,1\,1\,1\,0\,1\,1\,0\,1\,1\,1\,]$$

De igual forma, se obtienen los conjuntos S' y S''

$$S' = [1,7],\ S'' = [0,4,8]$$

A la solución $y_{\cap}$ se le agregan los proveedores contenidos en S', de esta manera se obtiene nuevamente y'. La Tabla 2.6 muestra los movimientos realizados.

Tabla 2.6. Provedores de S' agregados a $y_{\cap}$.

Proveedor	$y_{\cap}$
$S' = 1$	[0 **1** 1 0 0 1 0 0 0 1]
$S' = 7$	[0 1 1 0 0 1 0 **1** 0 1]

En y' se eliminarán los proveedores contenidos en el conjunto S' en las iteraciones impares y se agregarán los proveedores contenidos en S'' en las iteraciones pares, para de esta manera obtener y''.

La tabla 2.7 muestra los movimientos necesarios para alcanzar dicho objetivo.

Tabla 2.7. Eliminar proveedores de S' y agregar proveedores de S''.

Iter	**Acción**	**Proveedor**	y'
1	Eliminar	$S' = 1$	[0 **0** 1 0 0 1 0 1 0 1]
2	Agregar	$S'' = 0$	[**1** 0 1 0 0 1 0 1 0 1]
3	Eliminar	$S' = 7$	[1 0 1 0 0 1 0 **0** 0 1]
4	Agregar	$S'' = 4$	[1 0 1 0 **1** 1 0 0 0 1]
5	Eliminar	---	[1 0 1 0 1 1 0 0 0 1]
6	Agregar	$S'' = 8$	[1 0 1 0 1 1 0 0 **1** 1]

Cabe hacer notar que en la iteración 5 no se hace ningún movimiento de eliminación ya que no existen más proveedores del conjunto S'. Como se aprecia en la tabla 2.7, se alternan los movimientos de eliminar y agregar proveedores de la solución.

Una vez que se ha alcanzado a y'', la última serie de movimientos consiste en agregar

los proveedores contenidos en el conjunto S' a y'' de esta manera se obtiene $y_{\cup}$. En la tabla 2.8 se observan los movimientos mencionados.

Tabla 2.8. Proveedores de S' agregados a y''.

Proveedor	$y_{\cup}$
$S'=1$	[1 **1** 1 0 1 1 0 0 1 1]
$S'=7$	[1 1 1 0 1 1 0 **1** 1 1]

2.6 Trabajos relacionados

Dado que el problema de abastecimiento internacional está muy relacionado con el problema de ubicación, 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 simple de ubicación de plantas, el problema capacitado de ubicación de la planta y una versión estocástica del problema. Sin embargo no reportan resultados [Kraup 1979].

Verter y Dincer, analizan el problema simple de ubicación de la plantas, la versión capacitada del problema, una versión estocástica y el problema de ubicación internacional de la planta. No mencionan ningún trabajo en problemas capacitados bajo incertidumbre [Verter 1992].

Louveaux y Peters, presentaron un problema basado en escenarios en el cual la capacidad de las plantas, es una decisión inicial, lo cual ciertamente se asemeja a limitar la capacidad de las plantas. Sin embargo, 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, consideran un solo producto, un solo periodo, además el precio y la demanda son inciertas. Presentan dos estrategias bajo la relación de precio/demanda. Una estrategia se basada en el precio y la producción. Asimismo presenta una estrategia que fija los precios con una producción óptima basada en vender hasta satisfacer la demanda [Jucker 1976].

Hodder y Jucker, consideran un único periodo, un único modelo y precios fijos. Incorporan precios correlacionados a un precio base. No consideran demanda precio elástica [Hodder 1982].

Hodder y Dincer, consideran ambas decisiones de ubicación y financiamiento simultáneamente. Toman en cuenta un solo periodo y un solo producto. Utilizan un modelo media-varianza y resaltan sus desventajas. Considera que todo el abasto, hasta un cierto límite, se venderá a una ganancia por unidad incierta [Hodder 1986].

Cohen y Lee, Exploran estrategias de manufactura que incluyen decisiones de ubicación de planta. Consideran un solo periodo, múltiples productos y no usan la demanda precio-elástica [Cohen 1989].

Gutierrez y Kuvelis, exploran la generación de escenarios para modelar la incertidumbre y resolver el problema simple de ubicación de plantas, lo cual equivale a un modelo de abastecimiento internacional con un criterio regret mínimax. Sin embargo, sus resultados deben ser examinados detalladamente, ya que las soluciones seleccionadas pueden ser peores a las soluciones del valor esperado en los casos en que se evalúan grandes conjuntos de escenarios. Tampoco evaluaron soluciones de un conjunto más grande de escenarios y solo reportaron el arrepentimiento máximo relativo en la muestra [Gutiérrez 1995].

Lawrence y Buss, consideran la capacidad de producción a nivel internacional. Plantean que la producción disgregada en varios países, se beneficia directamente con las fluctuaciones de sus monedas. Consideran la producción de un único tipo de producto en dos países, donde cada uno tiene su propia demanda de ese producto. Incluyen una sección para el modelado de las tasas de cambio. El tipo de cambio monetario es el único factor bajo incertidumbre [Lawrence 1999].

La Tabla 2.9 muestra un resumen de las principales características de los trabajos.

Tabla 2.9. Resumen de artículos que tratan el problema de ubicación de plantas.

Autores	Enfoque Robusto	Periodo	Producto	Capacidad
Louveraux y Peters, 1992	Escenarios			Infinita
Jucker y Carlson, 1976	Media varianza	Único	Único	Infinita
Hodder y Jucker, 1982,1985ª,1985b	Escenarios	Único	Único	
Haug, 1985		Múltiple	Único	
Hodder y Dincer, 1986	Media varianza	Único	Único	
Cohen y Lee, 1989		Único	Múltiple	
Gutierrez y Kouvelis, 1995	Criterio Minimax de arrepentimiento			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. Basan su enfoque robusto en el criterio minimax de pérdida. Entre las ventajas de este enfoque están el hecho de que no es necesario asignar probabilidades a los escenarios potenciales, además su función de costo es relativamente fácil de resolver. La desventaja es su punto de vista pesimista, debido a que se selecciona un escenario difícil de ocurrir. Por lo tanto, las soluciones buenas en la mayoría de los escenarios se descartan para dar paso a alguna solución que solo es buena en cierto escenario extremo [1Kouvelis 1997].

Escudero, presentó un trabajo relacionado con el abastecimiento externo en manufactura. Utiliza el costo esperado como el único termino a optimizar en la función objetivo [Escudero 1993].

Sobre el problema robusto del abastecimiento internacional con capacidad finita (ROCIS), se conocen los siguientes trabajos:

Laguna y Velarde, formulan el problema robusto de capacidad finita de abastecimiento internacional. Incluyen una sección donde se explica la generación de las instancias de prueba. Utilizan la estrategia de memoria a corto plazo correspondiente al método de búsqueda tabú para la navegación en el espacio de soluciones. Consideran la capacidad finita en la producción. Incluyen también la incertidumbre de las tasas de demanda e intercambio bajo una medida de riesgo y la modelan por medio de un conjunto de escenarios. Utilizan una variación de la optimización robusta para formular el problema planteado. Su función de riesgo solo penaliza las desviaciones indeseables. Utilizan los valores duales asociados a los proveedores para guiar mejor, hacia una óptima solución. Su conjunto de instancias no incluye los valores óptimos encontrados. Evalúan todos los movimientos factibles generados. Solo reportan promedios globales [González 2004].

Martí-Velarde, se basan en el artículo de Laguna y Velarde. Implementan el uso de re-encadenamiento de trayectorias en vez de la búsqueda tabú. Mejoran la solución inicial, implementando una medida de atractividad. Su conjunto de instancias no incluye los valores óptimos encontrados. Solo reportan promedios globales [González 2006].

En [Gómez 2007] 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. Los resultados experimentales muestran que una de las estrategias propuestas logran reducir en un 2.13% el consumo de recursos requeridos para resolver las instancias y en un 23% 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 11.43%. Dado lo alentador de los resultados, actualmente se está trabajando en la aplicación de estas estrategias en la mejora del desempeño de la solución de ROCIS que utiliza re-encadenamiento de trayectorias.

La Tabla 2.10 muestra las características relevantes de los trabajos anteriores.

Tabla 2.10. Resumen de artículos que tratan el problema de ubicación de plantas.

Artículo	**Estrategia de generación de soluciones iniciales**	**Estrategia de búsqueda local**
Laguna y Velarde, 2004	$G_i = \frac{f_i}{b_i}$	Búsqueda Tabú.
Marti y Velarde, 2006	$G_i = \frac{f_i + \left(\sum_{s \in S} p_s \left(\sum_{j=1}^{n} c_{ij} e_{is} \right) \right)}{b_i}, \forall j \in N, i \in M$	Re-encadenamiento de trayectorias
Gómez Carpizo	$G_i = \frac{f_i + \left(\sum_{s \in S} p_s \left(\sum_{j \in P_i^+} c_{ij} e_{is} \right) \right)}{b_i}, \quad \forall i \in M$ en donde $P_i^+ = \{ j \in N \mid c_{ij} < CF_i \}$ es el conjunto de plantas hacia las que al proveedor *i* le resulta más económico enviar producto	Búsqueda Tabú

Como se puede observar, las soluciones reportadas del problema ROCIS consisten básicamente consisten en la generación de un conjunto de soluciones iniciales de alta calidad y un proceso de búsqueda local generalizada en la vecindad de dichas soluciones, dos problemas abiertos consisten en tratar de mejorar cada uno de estos procesos. En esta tesis se aborda el problema de mejorar la solución del problema ROCIS en el contexto del proceso de búsqueda local.

Capítulo 3

PROPUESTA DE SOLUCIÓN

Este capítulo describe a detalle cada una de las propuestas por esta tesis para dar solución al problema ROCIS. La primera de ellas consiste en una mejora a la generación de la vecindad denominada Vecindad Mejorada. La segunda estrategia planteada es un Re-encadenamiento de Trayectorias que incluye dos estrategias. Finalmente se evalúa una estrategia que consiste en la suma de esfuerzos de las dos primeras propuestas de solución mencionadas anteriormente.

3.1 Búsqueda Tabú

Búsqueda Tabú (BT) es un procedimiento metaheurístico utilizado para guiar un algoritmo heurístico de búsqueda local para explorar el espacio de soluciones más allá de la simple optimalidad.

BT se basa en la premisa de que para poder calificar de inteligente la solución de un problema, debe incorporar memoria adaptativa y exploración sensible (responsive).

El énfasis de la exploración sensible en búsqueda tabú, ya sea una implementación determinística o probabilística, se deriva de la suposición de que una mala elección estratégica puede producir más información que una buena elección al azar.

Las características principales de la BT se describen a continuación.

3.1.1 Memoria Adaptativa

La memoria adaptativa en búsqueda tabú permite la implementación de procedimientos capaces de realizar la búsqueda en el espacio de soluciones eficaz y eficientemente.

Dado que las decisiones locales están por tanto guiadas por información obtenida a lo largo del proceso de búsqueda, la búsqueda tabú contrasta con diseños que por contra confían en procesos semialeatorios, que implementan una forma de muestreo. La memoria adaptativa también contrasta con los típicos diseños de memoria rígidos tales como las estrategias de ramificación y acotación. A continuación se muestran las características de la memoria adaptativa.

- Selectividad (incluyendo olvido estratégico)
- Asbtracción y descomposición (a través de memoria explícita y por atributos)
- Tiempo:
    - Recencia de eventos
    - Frecuencia de eventos
    - Diferenciación entre corto y largo plazo

- Calidad e impacto:
    - Atracción relativa de elecciones alternativas
    - Magnitud de cambios en relaciones de estructura o restricciones
- Contexto:
    - Interdependencia regional
    - Interdependencia estructural
    - Interdependencia secuencial

3.1.2 Exploración sensible

Consiste en la imposición estratégica de limitaciones; un enfoque concentrado en buenas regiones y buenas características de las soluciones, caracterización y exploración de nuevas regiones prometedoras, patrones de búsqueda no monótonos e integración y extensión de soluciones. A continuación se mencionan las características de la exploración sensible.

- Imposición estratégica de limitaciones e inducciones (*condiciones tabú y niveles aspiración*)
- Enfoque concentrado en buenas regiones y buenas características de las soluciones (*procesos de intensificación*)
- Caracterización y exploración de nuevas regiones prometedoras (*procesos de diversificación*)
- Integración y extensión de soluciones (*re-encadenamiento de trayectorias*)

La finalidad de la búsqueda tabú es encontrar formas nuevas y más efectivas de sacar ventaja a los conceptos descritos anteriormente e identificar principios asociados que puedan emplear los fundamentos de la búsqueda inteligente. A medida que esto sucede, nuevas mezclas estratégicas de las ideas básicas emergen, conduciendo a mejores soluciones y mejores implementaciones prácticas. Esto hace de BT un área fértil para la investigación y el estudio empírico.

3.1.3 Arquitectura general del algoritmo Tabú

Una vez que una solución se ha elegido como un punto de partida, la búsqueda tabú se inicia. Existen dos fases en la búsqueda que interactúan como se muestra en la Figura 3.1. Cada fase comienza en la solución actual y después de terminar regresan una mejor solución total y una nueva solución actual. La búsqueda termina cuando el número de iteraciones *maxGlobal* ha transcurrido sin mejorar la mejor solución total. Una iteración consiste en ejecutar los pasos 3 y 4. El paso 3 corresponde al proceso de intensificación y el paso 4 al proceso de diversificación.

La Figura 3.2 describe la fase de intensificación. Primero se declara como mejor solución actual a la solución actual. La solución actual es, ya sea la solución inicial generada con la construcción heurística (al inicio de la primera iteración global) o la última solución visitada después de la terminación de la fase de diversificación (después de la primera iteración global). La selección de una solución en el paso 2 es un proceso probabilístico que usa el concepto de valores de conteo introducidos por [Laguna 1999]. La búsqueda para la mejor posición en el paso 3 de la Figura 3.2 es exhaustiva. Por último, el paso 4 realiza las tareas de actualización.

```
1. Encontrar una solución inicial p con la construcción heurística
2. Actualizar la mejor solución total encontrada (poverall_best = p)
mientras (número de iteraciones globales sin mejorar el major total <
maxGlobal)
{
    3. Ejecutar la fase de intensificación iniciando desde la solución
       actual p. Regresa la mejor solución total p_overall best y la
       solución actual p.
    4. Ejecutar la fase de diversificación iniciando desde la solución
       actual p. Regresa la mejor solución total p_overall best y la
       solución actual p.
}
```

Figura 3.1. Arquitectura general del algoritmo tabú.

1. Inicializar las estructuras de memoria tabú para poner todas las soluciones en el estado de "*no tabú activo*" y borrar todos los contadores de frecuencia. Hacer la solución actual la mejor solución actual ($p_{current\ best}=p$)

Mientras (iteraciones locales sin mejorar el mejor actual < *maxIntensify*)

{

2. Elegir una solución en estado tabú no activo. La probabilidad de elegir una solución es inversamente proporcional a su valor de conteo.
3. Mover la solución elegida a la mejor posición que es diferente de la posición actual, inclusive si el valor de la función objetivo incrementa. La solución actual *p* es la que resulta después del movimiento.
4. Hacer a la solución seleccionada "*tabú activo*", y si es apropiado, actualizar $p_{current\ best}$ y/o $p_{overall\ best}$. También, actualizar las estructuras de memoria tabú para liberar las soluciones del estado *tabú activo* e incrementar sus contadores de frecuencia.

}

Figura 3.2. Fase de intensificación del algoritmo tabú.

3.2 Solución Tabú

El algoritmo de búsqueda tabú reportado en [González 2004], en el que se basa la propuesta de este trabajo, las estrategias que se proponen se describen en las secciones 3.3, 3.4 y 3.5 de este capítulo. El algoritmo tabú base se describe a continuación.

3.2.1 Algoritmo de la solución tabú [González 2004]

La solución Tabú consiste de 3 etapas: la primera de ellas es la *Solución Inicial*, en la cual se calcula una primera solución que servirá como punto de partida para generar nuevas soluciones. La segunda etapa es la de *Generación de Vecinos*, en ella se generan soluciones

vecinas a partir de la solución inicial. Una vez que se han generado los vecinos, tiene lugar la tercera etapa que consiste en la *Búsqueda Local*, ahí se hace realiza una búsqueda exhaustiva de las soluciones generadas en la segunda etapa.

Una vez que ha concluido la tercera etapa se regresa a la segunda etapa con la mejor solución obtenida durante el proceso y se realiza un número de iteraciones que se determina desde el inicio del proceso.

La Figura 3.3 muestra la estructura de la solución tabú.

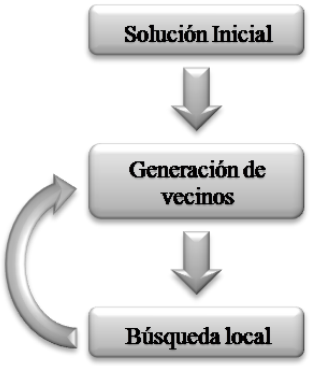

Figura 3.3. Estructura de la solución Tabú.

Las estructuras utilizadas son las siguientes:

- Selección de proveedores: $y = y_1, y_2, ..., y_m$
- Representación interna: $H(y) = \sum_{i \in M} y_i 2^i$
- Listas proveedores tabú: *inserción, eliminación e intercambio*.

La Tabla 3.1 describe el algoritmo de la solución tabú.

Tabla 3.1. Algoritmo de la solución tabú reportada en [González 2004].

Línea	Descripción
1	1. Cargar datos de la instancia
2	2. Generar la solución inicial factible *y*

a. Calcular $D = \max_{s \in S}(\sum_{j \in N} d_{js})$

b. Ordenar los proveedores de manera ascendente por $G_i = \frac{f_i}{b_i}$

c. 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*.

d. Determinar *F*[*y*], resolviendo los subproblemas de distribución que se generan en todos los escenarios.

3. Hacer *n* iteraciones

a. Determinar el valor r_i de cada proveedor.

i. Determinar el valor esperado de los precios sombra (π_{is}) asociados a la restricción correspondiente a cada proveedor, en la solución de los subproblemas asociados a la solución *y* en todos los escenarios posibles.

$$E(\pi_i) = \sum_{s \in S} p_s \pi_{is}$$

ii. Calcular el costo relativo de los proveedores (r_i.)

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

b. 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 *D*.

c. Se integran las listas de movimientos candidatos de inserción, eliminación e intercambio, a partir de los movimientos identificados en el paso anterior si los proveedores involucrados no forman parte de la lista tabú correspondiente al tipo de movimiento.

i. La lista de candidatos de inserción contiene los 3 proveedores de menor valor r_i.

ii. La lista de candidatos de eliminación contiene los 3 proveedores de mayor valor r_i.

iii. La lista de candidatos 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 configuración *y*.

d. Para cada configuración *y*' generada a partir de los movimientos de las listas de candidatos de inserción, eliminación e intercambio:

20 i. Calcular el valor de la función objetivo

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

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

21 ii. 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 durante las que se considera tabú a los proveedores involucrados en el movimiento es $\frac{m}{3}$ para inserciones y eliminaciones y $\frac{m(m-1)}{16}$ para intercambios.

22 iii. Actualizar la mejor solución encontrada.

23 iv. Hacer $y = y_{mejor}$

24 4. Regresa la mejor solución encontrada

3.3 Propuesta de solución 1: Vecindad Mejorada (VM)

En la etapa de la *Generación de Vecinos* es en donde se ha propuesto una solución para mejorar específicamente el cálculo de los costos relativos del proveedor *i* (r_i). El costo relativo para el proveedor *i* se calcula como se muestra en (3.1):

$$r_i = \begin{cases} \frac{E(\pi_i)}{f_i} & si \quad E(\pi_i) < 0 \\ f_i & si \quad E(\pi_i) = 0 \end{cases} \qquad (3.1)$$

donde f_i es el costo fijo del proveedor *i* y $E(\pi_i)$ es el precio dual esperado del proveedor *i*, cuyo cálculo se presenta en (3.2)

$$E(\pi_i) = \sum_{s \in S} p_s \pi_{is} \qquad (3.2)$$

donde p_s es la probabilidad de que ocurra el escenario *s* y π_{is} es el precio dual del proveedor *i* en el escenario *s*. El precio dual del proveedor *i* se obtiene de los precios duales resultantes de resolver el subproblema de transporte asociado a cada escenario. En el contexto actual, se examinan los precios duales correspondientes a las restricciones de capacidad en cada subproblema de un escenario.

Dichos costos relativos intervienen en la generación de las listas de candidatos para inserción, borrado e intercambio. Aquellos proveedores con el costo relativo más pequeño formarán la lista de candidatos de inserción. Los proveedores que tengan el costo relativo de mayor valor formarán la lista de candidatos de eliminación. Ya que un intercambio consiste en reemplazar un proveedor *i* actualmente seleccionado con un proveedor *j* actualmente no seleccionado, se define el costo relativo de un intercambio como $r_j - r_i$ y se incluyen en la lista de candidatos de intercambio a aquellos intercambios con los valores más pequeños. La Tabla 3.2 muestra de manera concentrada como se forma la lista de candidatos.

Tabla 3.2. Criterio para formar las listas de candidatos.

Lista	**Proveedores**
Inserción	Menor r_i
Eliminación	Mayor r_i
Intercambio	Menor diferencia $r_j - r_i$

3.3.1 Análisis de r_i

Teniendo en cuenta a (4.3)

$$r_i = \frac{E(\pi_i)}{f_i}, \qquad E(\pi_i) < 0 \tag{3.3}$$

se observa que, el costo relativo r_i es mayor sí y sólo sí $E(\pi_i)$ es mayor y/o f_i es mayor. De manera equivalente, el costo relativo r_i es menor sí y sólo sí $E(\pi_i)$ es menor y/o f_i es menor.

3.3.2 Interpretación de la inserción y la eliminación

Eliminación: se eliminan aquellos proveedores para los cuales el costo relativo r_i es mayor. Esto significa que se eligen aquellos proveedores para los cuales $E(\pi_i)$ es mayor (incrementan menos la función objetivo por unidad de recurso disponible) y/o su costo fijo f_i es mayor. Se eliminan los proveedores con respecto a los cuales la función objetivo es menos sensible.

Inserción: se insertan aquellos proveedores para los cuales su costo relativo r_i es menor. Esto significa que se eligen aquellos proveedores para los cuales $E(\pi_i)$ es menor (incrementan más la función objetivo por unidad de recurso disponible) y/o su costo fijo f_i es menor. Se insertan los proveedores con respecto a los cuales la función objetivo es más sensible.

Una limitante de este enfoque es que sólo se considera para determinar la sensibilidad de la función objetivo solo se consideran el valor esperado de los precios duales. Por lo tanto si para un proveedor $E(\pi_i) = -10,\ f_i = 3,\ b_i = 10$, y para un segundo proveedor se tiene que $E(\pi_i) = -20,\ f_i = 3,\ b_i = 2$; entonces bajo este criterio se elige para inserción al segundo proveedor y para eliminación al primero. Esto se debe a que en el supuesto de que ambos tengan el mismo costo fijo se elige en un caso al de menor $r_i = -20/3$ y para eliminación al de mayor $r_i = -10/3$. Sin embargo esto sólo refleja el hecho de que la tasa de incremento de la capacidad de cada de los proveedores es mayor en el primer caso que en el segundo.

Por otra parte si se considera el incremento que sufriría la función objetivo, en el

primer caso este es de $\left|E(\pi_i)\cdot b_i\right|=100$ y en el segundo caso es de $\left|E(\pi_i)\cdot b_i\right|=40$. Bajo este criterio la función objetivo es más sensible (incrementa más) para el primer proveedor y menos sensible para el segundo. Como se puede observar la sensibilidad de la función objetivo depende simultáneamente de $E(\pi_i)$ y b_i, lo cual es consistente con el hecho de que cuando se selecciona un proveedor en la restricción correspondiente mostrada en (3.4)

$$\sum_{i\in N} x_{ijs} \le b_i y_i \,, \quad \forall i \in M \tag{3.4}$$

el lado derecho de ésta toma el valor de b_i.

3.3.3 Solución Propuesta

La solución propuesta consiste en modificar la expresión de r_i para que la elección de los proveedores considere el incremento absoluto de la función objetivo ($\left|E(\pi_i)\cdot b_i\right|$) en lugar de sólo considerar la tasa de incremento ($\left|E(\pi_i)\right|$). Esto se logra multiplicando la capacidad del proveedor *i* por el costo relativo; quedando la obtención de los r_i de la siguiente manera como se muestra en (3.5):

$$r_i = \begin{cases} \dfrac{E(\pi_i)\cdot b_i}{f_i} & si \quad E(\pi_i)<0 \\ f_i & si \quad E(\pi_i)=0 \end{cases} \tag{3.5}$$

3.3.4 Análisis de r_i

Teniendo en cuenta a la expresión (3.6):

$$r_i = \frac{E(\pi_i)\cdot b_i}{f_i}\cdot, \qquad E(\pi_i) < 0 \qquad (3.6)$$

se observa que, el costo relativo r_i es mayor sí y sólo sí $E(\pi_i)\cdot b_i$ es mayor y/o f_i es mayor. De manera equivalente, el costo relativo r_i es menor sí y sólo sí $E(\pi_i)\cdot b_i$ es menor y/o f_i es menor.

3.3.5 Interpretación de la inserción y la eliminación

Eliminación: se eliminan aquellos proveedores para los cuales su costo relativo r_i es mayor. Esto significa que se eligen aquellos proveedores para los cuales $E(\pi_i)\cdot b_i$ es mayor (el incremento de la función objetivo $\left|E(\pi_i)\cdot b_i\right|$ es menor) y/o su costo fijo f_i es mayor. Se eliminan los proveedores con respecto a los cuales la función objetivo es menos sensible.

Inserción: se insertan aquellos proveedores para los cuales su costo relativo r_i es menor. Esto significa que se eligen aquellos proveedores para los cuales $E(\pi_i)\cdot b_i$ es menor (el incremento de la función objetivo $\left|E(\pi_i)\cdot b_i\right|$ es mayor) y/o su costo fijo f_i es menor. Se insertan los proveedores con respecto a los cuales la función objetivo es más sensible.

3.4 Propuesta de solución 2: Re-encadenamiento de Trayectorias (RT)

La segunda propuesta de solución Re-encadenamiento de Trayectorias descrita en la sección 2.5 tiene lugar después de la tercera etapa de la estructura de la solución tabú que es *"Búsqueda Local"*.

El algoritmo de *Re-encadenamiento de trayectorias* se describe a continuación:

4. Para cada par de soluciones y' y y'' de proveedores.
    a. Calcular dos nuevas soluciones $y_\cap$ y $y_\cup$ donde $y_{\cap i} = 1$ sí $y'_i = 1$ y $y''_i = 1$, 0 en caso contrario, $y_{\cap i} = 1$ sí $y'_i = 1$ ó $y''_i = 1$, 0 en caso contrario.

b. Calcular los conjuntos S' (proveedores seleccionados en y' pero no seleccionados en y'') y S'' (proveedores no seleccionados en y' y seleccionados en y'').

c. Agregar a la solución $y_{\cap}$, los proveedores del conjunto S' de uno en uno en el orden dado hasta alcanzar a y'.

d. Una vez alcanzada y' se alternará entre eliminar en las iteraciones impares a un proveedor de S' y agregar a un proveedor de S'' en las iteraciones pares hasta alcanzar a y''.

e. Agregar a la solución y'', los proveedores del conjunto S' de uno en uno en el orden dado hasta alcanzar a $y_{\cup}$.

f. Evaluar todas las soluciones resultantes y guardar aquellas que obtengan un mejor valor que y' y y''.

5. Devolver la solución con el mejor valor.
6. Fin

Una vez que se ha conseguido obtener una nueva solución en la etapa de búsqueda local, a continuación se guarda dicha solución con la condición de que sea mejor que la anterior. Es decir, durante el proceso se guardan las dos mejores soluciones para posteriormente aplicar el re-encadenamiento de trayectorias entre las soluciones obtenidas. Al realizar el re-encadenamiento de trayectorias se hace una búsqueda entre las soluciones generadas para alcanzar nuevas y mejores soluciones con lo cual se pueda llegar a la solución óptima en un menor número de iteraciones.

3.4.1 Estrategias de re-encadenamiento de trayectorias

En esta propuesta de solución se han considerado dos estrategias para aplicar el re-encadenamiento de trayectorias a dos soluciones obtenidas durante el proceso.

3.4.1.1 Estrategia 1

Durante el proceso de solución mostrado en la Figura 3.1 se realiza un número determinado de iteraciones; cada iteración considera las etapas 2 y 3 de la estructura de la solución tabú. Al final de cada iteración se obtiene una mejor solución. La estrategia 1 consiste en realizar el re-encadenamiento de trayectorias al final de cada iteración una vez que se ha obtenido una mejor solución, es decir; la propuesta de solución se lleva a cabo con las 2 mejores soluciones que se han obtenido al final de cada iteración. La Figura 3.2 muestra el pseudocódigo de la función *search* que es donde se realiza el re-encadenamiento.

3.4.1.2 Estrategia 2

La estrategia 2 del re-encadenamiento de trayectorias se lleva a cabo una vez que se ha finalizado el ciclo de iteraciones que considera las etapas 2 y 3 de la solución tabú. En cada iteración se guarda la nueva solución encontrada si cumple la condición de ser mejor que la anterior. De esta manera al final del ciclo se cuenta con las 2 mejores soluciones obtenidas durante todo el proceso para realizar el re-encadenamiento de trayectorias con ellas. La Figura 3.4 muestra el lugar en donde tiene lugar la estrategia 2 del re-encadenamiento de trayectorias.

```
Inicio
...
Mientras iter <= maxiter
      form_list
      select move
      execute_move
      actualiza las 2 mejores soluciones
      //path_relinking( ) - RT Estrategia 1
Fin mientras
// path_relinking( ) - RT Estrategia 2
Fin
```

Figura 3.4. Pseudocódigo de la función *search*.

3.5 propuesta de solución 3: Vecindad Mejorada + Re-Encadenamiento de Trayectorias (VM + RT)

La tercera propuesta de solución es una suma de esfuerzos entre las primeras dos estrategias. Consiste en aplicar la vecindad mejorada (VM) y el re-encadenamiento de trayectorias (RT) de

manera conjunta. Primero se ejecuta VM ya que tiene lugar en la segunda etapa del proceso de solución que es *"generación de vecinos"* y posteriormente RT que opera en la tercera etapa de la solución *"búsqueda local"*. En cada iteración operan de manera conjunta VM y RT.

Es importante mencionar que en esta tercera propuesta de solución el re-encadenamiento de trayectorias RT también se lleva a cabo con las 2 estrategias descritas en las secciones 3.3.1.1 y 3.3.1.2. Así, la tercera propuesta de solución queda descrita de la siguiente manera:

- Vecindad mejorada + Re-encadenamiento de trayectorias Estrategia 1 (VM + RT E1).
- Vecindad mejorada + Re-encadenamiento de trayectorias Estrategia 2 (VM + RT E2).

Capítulo 4

RESULTADOS EXPERIMENTALES

4.1 Plataforma Experimental

Las estrategias de mejora propuestas en este trabajo fueron implementadas sobre el código reportado en [Gómez 2007]. Las instancias utilizadas durante la experimentación son las reportadas en [González 2004], con estas se midió el desempeño de las estrategias de mejora. Estas instancias consideran 10 plantas y 27 escenarios en cada una de ellas. El total de 90 instancias se dividen en 3 grupos de 30 instancias, considerando 10, 15 y 20 proveedores. La solución óptima de dichas instancias se obtuvo mediante un método exhaustivo.

Para llevar a cabo lo experimentos, se utilizó una computadora DELL OPTIPLEX 1601 con un procesador Intel Pentium IV a 2.4 GHZ y una capacidad de memoria RAM de 1 GB. El código utilizado reportado en [Gómez 2007] se compiló en Visual C 6.0, el sistema operativo fue Windows Xp Profesional Service Pack 2. Para dar solución al subproblema de transporte de ROCIS se utilizó LINDO API 2.0.

Es importante mencionar que los resultados obtenidos al realizar los experimentos que se describen posteriormente, se mostrarán en tablas que cuentan con las siguientes características (descritas por columna):

1. Cantidad de proveedores del grupo analizado.
2. Iteraciones promedio hasta encontrar el óptimo.
3. Tiempo promedio hasta encontrar el óptimo.
4. Tiempo promedio en procesar la instancia.
5. Número de óptimos alcanzados.

Al final de cada columna se muestra los valores acumulados de cada una de las características mencionadas anteriormente.

Con el objetivo de medir el desempeño de cada una de las estrategias de mejora propuestas en este trabajo, se evalúan la calidad como primer indicador del desempeño, y la eficiencia de las mismas. Al evaluar la calidad de la solución de las estrategias propuestas se considera el número total de óptimos alcanzados del total de las 90 instancias.

Para la evaluación de la eficiencia de la solución de cada una de las estrategias se consideraron los siguientes indicadores:

1. Iteraciones promedio hasta encontrar el óptimo.
2. Tiempo promedio hasta encontrar el óptimo.
3. Tiempo promedio en procesar la instancia.

4.2 Forma en que se obtienen los resultados

La forma en que se obtienen los resultados se describe a continuación. Una vez que se tiene el

archivo con los resultados (Tabla 4.1), en este caso se muestran los resultados para el grupo de 30 instancias de 10 proveedores con 10 iteraciones de la solución de referencia reportada en [González 2004], se constituyen las tablas de resultados descritas al inicio de esta sección.

Tabla 4.1. Resultados para grupo de 30 instancias de 10 proveedores, 10 iteraciones [González 2004].

Tam	Instancia	H(y) Inicial	F(y) inicial	H(y) Final	y	Iter hasta el óptimo	Tiempo hasta el óptimo	Tiempo de instancia	F(y) Final
10	rocis_10_10_0.6_0.5_1.txt	935	44217.568500	994	1111100010	2	1.81	4.78	37728.921125
10	rocis_10_10_0.6_0.5_2.txt	414	50312.625797	755	1011110011	4	3.64	8.09	32214.637019
10	rocis_10_10_0.6_0.5_3.txt	813	52929.665349	478	0111011110	4	2.78	5.47	35353.188673
10	rocis_10_10_0.6_0.5_4.txt	945	29384.575562	956	1110111100	2	1.78	5.14	24966.885889
10	rocis_10_10_0.6_0.5_5.txt	702	46217.704716	883	1101110011	3	2.88	6.01	38164.969340
10	rocis_10_10_0.6_1.0_6.txt	819	46760.400521	222	0011011110	4	3.41	7.23	34679.363356
10	rocis_10_10_0.6_1.0_7.txt	947	46107.595453	502	0111110110	2	1.73	5.03	32939.743241
10	rocis_10_10_0.6_1.0_8.txt	373	68247.090017	655	1010001111	4	3.47	6.99	51165.003695
10	rocis_10_10_0.6_1.0_9.txt	63	57319.916352	757	1011110101	3	2.64	6.03	50267.772910
10	rocis_10_10_0.6_1.0_10.txt	994	50568.955540	691	1010110011	2	1.66	6.27	46096.084502
10	rocis_10_10_0.6_2.0_11.txt	461	62862.990153	63	0000111111	3	2.59	6.67	51547.311451
10	rocis_10_10_0.6_2.0_12.txt	933	76025.175146	380	0101111100	3	2.64	6.20	58898.443603
10	rocis_10_10_0.6_2.0_13.txt	1000	80242.333933	183	0010110111	4	3.39	6.67	66555.081865
10	rocis_10_10_0.6_2.0_14.txt	691	90835.946969	489	0111101001	3	2.48	5.84	66908.380268
10	rocis_10_10_0.6_2.0_15.txt	966	63612.533545	474	0111011010	2	1.78	5.23	56987.982465
10	rocis_10_10_0.3_0.5_16.txt	193	60316.264511	858	1101011010	6	4.76	7.53	28912.820977
10	rocis_10_10_0.3_0.5_17.txt	176	63976.269951	78	0001001110	4	3.11	7.00	43671.590564
10	rocis_10_10_0.3_0.5_18.txt	593	53002.180698	665	1010011001	3	2.69	8.05	38462.455319
10	rocis_10_10_0.3_0.5_19.txt	137	58647.417282	296	0100101000	4	3.20	5.42	39953.401589
10	rocis_10_10_0.3_0.5_20.txt	11	51173.715653	204	0011001100	3	2.38	7.81	36739.459480
10	rocis_10_10_0.3_1.0_21.txt	417	97607.098638	553	1000101001	2	1.83	7.95	67531.821818
10	rocis_10_10_0.3_1.0_22.txt	676	57770.948492	170	0010101010	2	1.64	6.34	49480.165911
10	rocis_10_10_0.3_1.0_23.txt	150	68964.967508	417	0110100001	3	2.50	8.13	45215.853605
10	rocis_10_10_0.3_1.0_24.txt	452	44549.208235	774	1100000110	2	1.69	6.80	33031.438185

10	rocis_10_10_0.3_1.0_25.txt	656	49562.370995	704	1011000000	1	0.86	5.34	41261.460220
10	rocis_10_10_0.3_2.0_26.txt	90	74006.472859	140	0010001100	3	2.75	7.08	52255.47197
10	rocis_10_10_0.3_2.0_27.txt	58	80000.835890	273	0100010001	3	2.47	6.14	60818.05733
10	rocis_10_10_0.3_2.0_28.txt	773	93943.303128	240	0011110000	4	3.39	7.75	67861.159615
10	rocis_10_10_0.3_2.0_29.txt	896	98707.540144	25	0000011001	3	2.22	6.30	79123.404253
10	rocis_10_10_0.3_2.0_30.txt	522	61534.475043	396	0110001100	4	2.89	6.63	54272.358103

La primer columna de la Tabla 4.1 muestra el tamaño de la instancia (en este caso 10 proveedores), la 2ª columna muestra la instancia; la 3ª columna muestra el valor inicial decimal de solución binaria *y* denotado por $H\ y$; la 4ª columna indica el $F\ y$ inicial, es decir, el valor de la función objetivo ROCIS; la 5ª columna muestra el valor de la solución binaria final; la 6ª columna muestra la solución binaria y ; la 7ª, 8ª y 9ª columnas indican el número de iteraciones hasta el óptimo, el tiempo hasta el óptimo y el tiempo en procesar la instancia respectivamente; por último la 10ª columna muestra el valor final de la función objetivo.

Con los resultados descritos anteriormente se obtienen datos de forma resumida que son el número promedio de iteraciones hasta encontrar el óptimo, el tiempo promedio en encontrar el óptimo, el tiempo promedio en procesar una instancia y el número de óptimos encontrados. La Tabla 4.2 muestra dichos resultados.

Tabla 4.2. Resultados con 10 iteraciones para grupo de 10 proveedores, [González 2004].

Proveedor	Instancia	# Iteraciones hasta encontrar el óptimo	Tiempo hasta encontrar el óptimo	Tiempo de instancia	Instancia resuelta
10	rocis_10_10_0.6_0.5_1.txt	2	1.81	4.78	1
10	rocis_10_10_0.6_0.5_2.txt	4	3.64	8.09	1
10	rocis_10_10_0.6_0.5_3.txt	4	2.78	5.47	1
10	rocis_10_10_0.6_0.5_4.txt	2	1.78	5.14	1
10	rocis_10_10_0.6_0.5_5.txt	3	2.88	6.01	1
10	rocis_10_10_0.6_1.0_6.txt	4	3.41	7.23	1
10	rocis_10_10_0.6_1.0_7.txt	2	1.73	5.03	1
10	rocis_10_10_0.6_1.0_8.txt	4	3.47	6.99	1
10	rocis_10_10_0.6_1.0_9.txt	3	2.64	6.03	1
10	rocis_10_10_0.6_1.0_10.txt	2	1.66	6.27	1
10	rocis_10_10_0.6_2.0_11.txt	3	2.59	6.67	1

Proveedor	Instancia	# Iteraciones hasta encontrar el óptimo	Tiempo hasta encontrar el óptimo	Tiempo de instancia	Instancia resuelta
10	rocis_10_10_0.6_2.0_12.txt	3	2.64	6.2	1
10	rocis_10_10_0.6_2.0_13.txt	4	3.39	6.67	1
10	rocis_10_10_0.6_2.0_14.txt	3	2.48	5.84	1
10	rocis_10_10_0.6_2.0_15.txt	2	1.78	5.23	1
10	rocis_10_10_0.3_0.5_16.txt	6	4.76	7.53	1
10	rocis_10_10_0.3_0.5_17.txt	4	3.11	7	1
10	rocis_10_10_0.3_0.5_18.txt	3	2.69	8.05	1
10	rocis_10_10_0.3_0.5_19.txt	4	3.2	5.42	1
10	rocis_10_10_0.3_0.5_20.txt	3	2.38	7.81	1
10	rocis_10_10_0.3_1.0_21.txt	2	1.83	7.95	1
10	rocis_10_10_0.3_1.0_22.txt	2	1.64	6.34	1
10	rocis_10_10_0.3_1.0_23.txt	3	2.5	8.13	1
10	rocis_10_10_0.3_1.0_24.txt	2	1.69	6.8	1
10	rocis_10_10_0.3_1.0_25.txt	1	0.86	5.34	1

Tabla 4.2. Continuación…

Proveedor	Instancia	# Iteraciones hasta encontrar el óptimo	Tiempo hasta encontrar el óptimo	Tiempo de instancia	Instancia resuelta
10	rocis_10_10_0.3_2.0_26.txt	3	2.75	7.08	1
10	rocis_10_10_0.3_2.0_27.txt	3	2.47	6.14	1
10	rocis_10_10_0.3_2.0_28.txt	4	3.39	7.75	1
10	rocis_10_10_0.3_2.0_29.txt	3	2.22	6.3	1
10	rocis_10_10_0.3_2.0_30.txt	4	2.89	6.63	1
	Promedio	**3.066666667**	**2.568666667**	**6.530666667**	**30**

Una vez que se tienen los promedios y el número de óptimos, se forman las tablas con los promedios para cada grupo de 30 instancias (10, 15 y 20 proveedores) como se muestra en la Tabla 4.3. Además, de esta manera se pueden obtener los acumulados de dichos promedios para de esta forma obtener una medida del desempeño de la estrategia de solución propuesta.

Tabla 4.3. Resultados con 10 iteraciones de la solución reportada en [González 2004].

Proveedor	Iteraciones promedio hasta encontrar el óptimo	Tiempo promedio hasta encontrar el óptimo	Tiempo promedio en procesar la instancia	Número de óptimos
10	3.066666667	2.568666667	6.530666667	30
15	3.966666667	7.808333333	17.55666667	28
20	5.266666667	19.60933333	34.76233333	27
Acumulado	**12.3**	**29.98633333**	**58.84966667**	**85/90**

4.3 Experimento 1: Análisis del desempeño de la estrategia vecindad mejorada (VM)

En este experimento se evalúa el desempeño de la estrategia de solución propuesta que consiste en la Vecindad Mejorada (VM) descrita en la sección 3.3.

4.3.1 Objetivo

Evaluar el desempeño de la estrategia de solución propuesta VM con respecto a la solución de referencia reportada en [González 2004].

4.3.2 Procedimiento

Para este experimento se utilizaron diferentes valores de alfa (α). El número de iteraciones utilizado en este experimento fueron de 10, 15 y 50 iteraciones. Se obtuvo el desempeño de la propuesta de solución VM para posteriormente hacer un comparativo con la solución de referencia.

4.3.3 Resultados

A continuación se muestran los valores acumulados de los resultados obtenidos por la solución propuesta VM. Las tablas 4.4, 4.5 y 4.6 muestran los resultados para 10, 15 y 50 iteraciones respectivamente utilizando en cada una de ellas los diferentes valores de α.

Tabla 4.4. Resultados acumulados para 10 iteraciones de la propuesta de solución VM.

α (alfa)	Iteraciones promedio hasta encontrar el óptimo	Tiempo promedio hasta encontrar el óptimo	Tiempo promedio en procesar la instancia	Número de Óptimos
0.1	9.73	23.52	52.52	86
0.2	10.23	24.29	52.65	86
0.3	10.16	24.69	53.05	86
0.4	9.87	23.8	52.41	88
0.5	10.47	24.83	52.37	86
0.6	9.87	23.61	51.55	88
0.7	9.63	22.52	51.72	85
0.8	9.17	21.86	51.66	86
0.9	9.63	23.04	52.31	83

Tabla 4.5. Resultados acumulados para 15 iteraciones de la propuesta de solución VM.

α (alfa)	Iteraciones promedio hasta encontrar el óptimo	Tiempo promedio hasta encontrar el óptimo	Tiempo promedio en procesar la instancia	Número de Óptimos
0.1	10.03	24.22	66.83	86
0.2	10.87	25.55	67.28	88
0.3	10.47	25.41	67.95	86
0.4	10.2	24.35	65.54	89
0.5	10.7	25.34	67.38	87
0.6	10.33	25.09	67.15	88
0.7	9.87	23.14	65.99	86
0.8	9.17	22.07	66.29	86
0.9	10.17	24.46	67.5	86

Tabla 4.6. Resultados acumulados para 50 iteraciones de la propuesta de solución VM.

α (alfa)	Iteraciones promedio hasta encontrar el óptimo	Tiempo promedio hasta encontrar el óptimo	Tiempo promedio en procesar la instancia	Número de Óptimos
0.1	11.57	27.18	114.97	89
0.2	10.87	25.74	115.70	88
0.3	12.93	29.51	118.64	89
0.4	11.3	26.62	115.20	89
0.5	12.2	27.44	118.24	89
0.6	11.5	28.02	118.05	90
0.7	11.53	26.27	115.95	88
0.8	10.97	25.62	116.01	89
0.9	11.7	27.02	117.33	88

4.3.4 Análisis de los resultados del experimento 1: VM

Como se puede apreciar en la Tabla 443, los mejores resultados para 10 iteraciones se obtienen con los valores de α=0.4 y α=0.6 ya que se logran 88 óptimos. Esto evaluando con respecto a la calidad de la solución (número de óptimos), pero si se evalúa con respecto a la eficiencia podemos observar que la solución que obtiene un mejor rendimiento es la de α=0.4 al tener un menor tiempo promedio en encontrar el óptimo y un menor tiempo promedio en procesar la instancia. Por lo tanto para 10 iteraciones, el mejor desempeño de la solución

propuesta VM se tiene con α=0.4.

Analizando la Tabla 4.5, el mejor resultado para 15 iteraciones se obtiene con un valor de α=0.4, ya que con este valor se obtiene el mayor número de óptimos, logrando 89 óptimos de los 90 posibles. En cuanto a la eficiencia, requiere de un mayor tiempo para alcanzar el número de óptimos obtenidos, pero aún así tiene mejor eficiencia con respecto a otros valores de α.

En la Tabla 4.6 notamos que el mejor resultado en cuanto a calidad es el que se obtiene con α=0.6 logrando los 90 óptimos posibles, es decir, que el conjunto de instancias de referencia mencionadas en la sección 5.1 se resuelve de manera óptima. En cuanto a la eficiencia se observa que se requiere de mayor tiempo para lograr el resultado obtenido pero sigue siendo mejor con respecto a otros valores de α, como α=0.3 y α=0.5.

Por lo tanto, podemos determinar que para la propuesta de solución VM el mejor resultado se obtuvo con 50 iteraciones y α=0.6 para de esta manera resolver de manera óptima las 90 instancias.

En la Figura 4.1 se muestra la gráfica comparativa del tiempo promedio hasta encontrar el óptimo para 10, 15 y 50 iteraciones. Dicha gráfica tiene como propósito mostrar el desempeño de la solución con respecto a la eficiencia. Como se puede apreciar en la gráfica, el tiempo promedio hasta encontrar el óptimo se incrementa conforme se aumenta el número de iteraciones. Para 10 y 15 iteraciones la diferencia es mínima, pero en el caso de 50 iteraciones aumenta de manera considerable. Para el valor de α=0.6 con 50 iteraciones la cual se determinó que obtiene el mejor rendimiento, se observa en la gráfica que es de las que más tiempo requiere para lograr el objetivo de alcanzar los 90 óptimos.

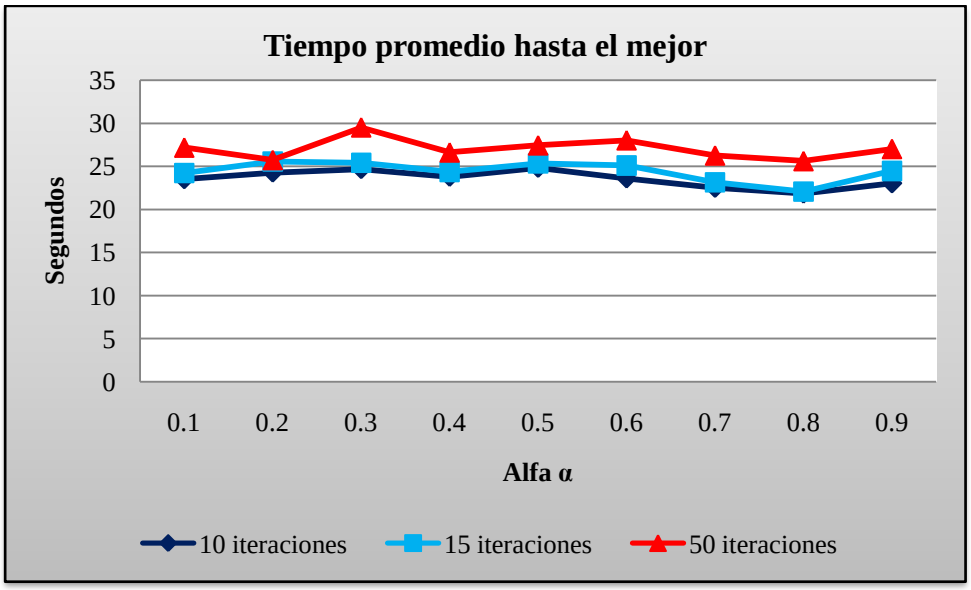

Figura 4.1. Gráfica comparativa para 10, 15 y 50 iteraciones de VM.

4.4 Experimento 2: análisis del desempeño de la estrategia re-encadenamiento de trayectorias (RT)

En este experimento se evalúa el desempeño de la propuesta de solución que consiste en el Re-encadenamiento de trayectorias (RT) con sus 2 estrategias (E1 y E2).

4.4.1 Objetivo

Evaluar el desempeño de la propuesta de solución RT con E1 y E2 con respecto a la solución de referencia reportada en [González 2004].

4.4.2 Procedimiento

Para este experimento se utilizaron diferentes valores de alfa (α). El número de iteraciones utilizado en este experimento fueron de 10, 15 y 50 iteraciones. Posteriormente se hará un comparativo con la solución de referencia reportada en [González 2004].

4.4.3 Resultados

A continuación se muestran los valores acumulados de los resultados obtenidos por la solución propuesta RT con sus 2 estrategias. Las tablas 4.7, 4.8 y 4.9 muestran los resultados para 10,

15 y 50 iteraciones respectivamente de E1. Las tablas 4.9, 4.10 y 4.11 muestran los resultados de E2para 10, 15 y 50 iteraciones respectivamente.

Tabla 4.7. Resultados acumulados para 10 iteraciones de la propuesta de solución RT con E1.

α (alfa)	**Iteraciones promedio hasta encontrar el óptimo**	**Tiempo promedio hasta encontrar el óptimo**	**Tiempo promedio en procesar la instancia**	**Número de Óptimos**
0.1	9.83	24.68	57.96	86
0.2	10.8	26.91	57.89	87
0.3	10.3	25.46	57.68	85
0.4	9.83	24.51	57.37	88
0.5	10.53	25.49	57.16	86
0.6	9.87	24.316	56.44	86
0.7	9.5	22.99	56.21	84
0.8	9.2	22.47	56.73	86
0.9	9.7	23.70	56.59	83

Tabla 4.8. Resultados acumulados para 15 iteraciones de la propuesta de solución RT con E1.

α (alfa)	**Iteraciones promedio hasta encontrar el óptimo**	**Tiempo promedio hasta encontrar el óptimo**	**Tiempo promedio en procesar la instancia**	**Número de Óptimos**
0.1	10.13	24.95	74.29	86
0.2	11.17	27.30	75.33	88
0.3	10.6	26.11	74.89	85
0.4	10.17	24.98	73.36	89
0.5	10.77	26.02	74.70	87
0.6	10.33	25.59	74.64	86
0.7	10.133	23.85	73.25	86
0.8	9.2	22.49	73.97	86
0.9	10.07	24.56	74.52	85

Tabla 4.9. Resultados acumulados para 50 iteraciones de la propuesta de solución RT con E1.

α (alfa)	Iteraciones promedio hasta encontrar el óptimo	Tiempo promedio hasta encontrar el óptimo	Tiempo promedio en procesar la instancia	Número de Óptimos
0.1	11.67	28.11	133.22	89
0.2	11.17	27.27	132.33	88
0.3	13.37	32.25	134.24	89
0.4	11.63	28.65	134.30	90
0.5	11.87	27.85	135.07	89
0.6	12.23	29.52	134.05	90
0.7	11.37	26.74	133.19	88
0.8	10.6	25.76	135.01	89
0.9	11.97	28.91	134.44	88

4.4.4 Análisis de los resultados del experimento 2: RT E1

En la Tabla 4.7 se observa que el mejor desempeño se obtiene con un valor de α=0.4 logrando 88 óptimos de los 90 posibles, esto al evaluar con respecto a la calidad de la solución. En cuanto a la eficiencia, requiere de una mayor cantidad de tiempo para resolver de manera óptima el mayor número de instancias, pero es aún mejor que para los valores de α=0.1, α=0.2 y α=0.3.

La Tabla 4.8 muestra que el desempeño más alto se obtuvo con un valor de α=0.4 al alcanzar 89 óptimos de los 90 posibles, siendo el mayor número de óptimos logrados con 15 iteraciones; esto con referencia a la calidad de la solución. Si se evalúa con respecto a la eficiencia y se toma como referente el tiempo promedio hasta encontrar el óptimo, podemos observar que para α=0.4 es la quinto mejor tiempo requerido al utilizar solamente 24.98 segundos, siendo superado por los valores de α=0.1, α=0.7, α=0.8 y α=0.9. Para α=0.8 con un tiempo promedio de 22.49 hasta encontrar el óptimo se obtiene el mejor resultado en cuanto a eficiencia, pero solamente logra 86 óptimos.

Para 50 iteraciones, la Tabla 4.9 muestra que el mejor desempeño con respecto a la calidad de la solución se obtiene con α=0.4 y α=0.6 logrando ambos valores el total de 90 óptimos. Evaluando con respecto a la eficiencia y tomando como referencia el tiempo promedio hasta encontrar el óptimo, para α=0.4 requiere de 28.65 segundos y para α=0.6 requiere de 29. 52 segundos, con lo cual se determina que el mejor es para el valor de α=0.4.

Después de analizar las tablas 4.7, 4.8 y 4.9 se determina que la mejor configuración de RT con E1 es para un valor de α=0.4 y 50 iteraciones ya que con esto se logra el total de 90 óptimos.

La Figura 4.2 muestra la gráfica comparativa de los tiempos promedio hasta encontrar el óptimo para 10, 15 y 50 iteraciones.

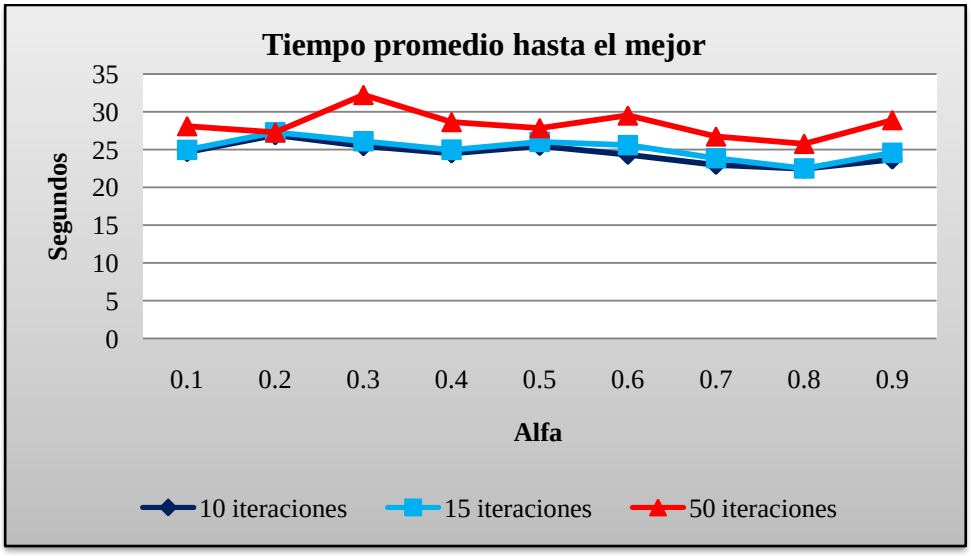

Figura 4.2. Gráfica comparativa para 10, 15 y 50 iteraciones de RT con E1.

Como se aprecia en la gráfica, para 10 y 15 iteraciones el tiempo requerido en segundos es muy similar, es decir, la diferencia entre ambos es muy pequeña. En el caso de 50 iteraciones se observa que el tiempo requerido se incrementa de manera considerable para el caso de α=0.3 y en cambio para α=0.2 es muy parecido para 10 y 15 iteraciones. Para el caso de α=0.4 y 50 iteraciones de RT con E1, la cual anteriormente se determinó que era la que mejor desempeño obtenía se observa un incremento considerable en el tiempo requerido para encontrar el óptimo.

Tabla 4.10. Resultados acumulados para 10 iteraciones de la propuesta de solución RT con E2.

α (alfa)	Iteraciones promedio hasta encontrar el óptimo	Tiempo promedio hasta encontrar	Tiempo promedio en procesar la instancia	Número de Óptimos

		el óptimo		
0.1	9.7	23.66	56.91	86
0.2	10.37	25.20	57.09	87
0.3	10.1	24.69	57.07	86
0.4	9.83	24.19	56.82	88
0.5	10.4	24.96	56.36	86
0.6	9.83	23.99	55.92	87
0.7	9.63	22.72	55.79	85
0.8	9.1	22.08	56.38	86
0.9	9.67	23.37	56.16	83

Tabla 4.11. Resultados acumulados para 15 iteraciones de la propuesta de solución RT con E2.

α (alfa)	Iteraciones promedio hasta encontrar el óptimo	Tiempo promedio hasta encontrar el óptimo	Tiempo promedio en procesar la instancia	Número de Óptimos
0.1	10	24.27	73.86	86
0.2	10.73	25.73	74.65	88
0.3	10.4	25.45	74.51	86
0.4	10.17	24.74	73.03	89
0.5	10.63	25.49	73.96	87
0.6	10.3	25.26	74.11	87
0.7	9.87	23.33	72.78	86
0.8	9.1	22.08	73.39	86
0.9	10.2	24.78	74.02	86

Tabla 4.12. Resultados acumulados para 50 iteraciones de la propuesta de solución RT con E2.

α (alfa)	Iteraciones promedio hasta encontrar el óptimo	Tiempo promedio hasta encontrar el óptimo	Tiempo promedio en procesar la instancia	Número de Óptimos
0.1	11.53	27.37	132.71	89
0.2	10.73	25.75	132.34	88
0.3	12.87	29.51	132.78	89
0.4	10.57	26.01	133.77	89
0.5	11.73	27.31	134.72	89
0.6	11.9	28.85	133.31	90
0.7	11.1	26.22	132.69	88
0.8	10.5	25.34	134.26	89
0.9	11.3	27.10	134.35	88

4.4.5 Análisis de los resultados del experimento 2: RT E2

En la Tabla 4.10 se observa que para10 iteraciones de RT con E2, para α=0.4 se obtiene el mayor número de óptimos al lograr 88 de los 90 posibles al evaluar con respecto a la calidad de la solución. Con respecto a la eficiencia, requiere de un mayor tiempo hasta encontrar el óptimo en comparación con valores de α=0.1, α=0.6, α=0.7, α=0.8 y α=0.9.

La Tabla 4.11 muestra los resultados obtenidos para 15 iteraciones de RT con E2. Evaluando la calidad de la solución se observa que para α=0.4 se logra el mayor número de óptimos con 89. En cuanto a la eficiencia, con α=0.4 requiere de menor tiempo para encontrar el óptimo en comparación con α=0.2, α=0.3, α=0.5, α=0.6 y α=0.9; pero su eficiencia es menor en comparación con α=0.1, α=0.7 y α=0.8 siendo estos últimos valores los que menor tiempo hasta encontrar en óptimo requieren.

Para 50 iteraciones de RT con E2, la Tabla 4.12 muestra que para el valor de α=0.6 se obtienen los 90 óptimos posibles al evaluar la calidad de la solución. Mientras que la eficiencia en tiempo es mayor para los otros valores de α, siendo realmente poca la diferencia con respecto a los demás valores de α. De esta se puede determinar que el mejor desempeño se obtiene con un valor de α=0.6 y 50 iteraciones.

La Figura 4.3 muestra la gráfica comparativa para 10, 15 y 50 iteraciones de RT con E2 del tiempo promedio hasta el mejor. Para el caso de 10 y 15 iteraciones se observa una diferencia mínima para casi todos los valores de α. En el caso de 50 iteraciones el tiempo requerido hasta encontrar el mejor se incrementa de forma considerable para los valores de α=0.3 y α=0.6. Para α=0.2 el tiempo requerido para los 3 casos de iteraciones es muy similar. Para los valores de α=0.6 y 50 iteraciones con los cuales se obtiene el mejor desempeño el tiempo requerido hasta el mejor es mayor que para los demás valores de α, excepto para α=0.3.

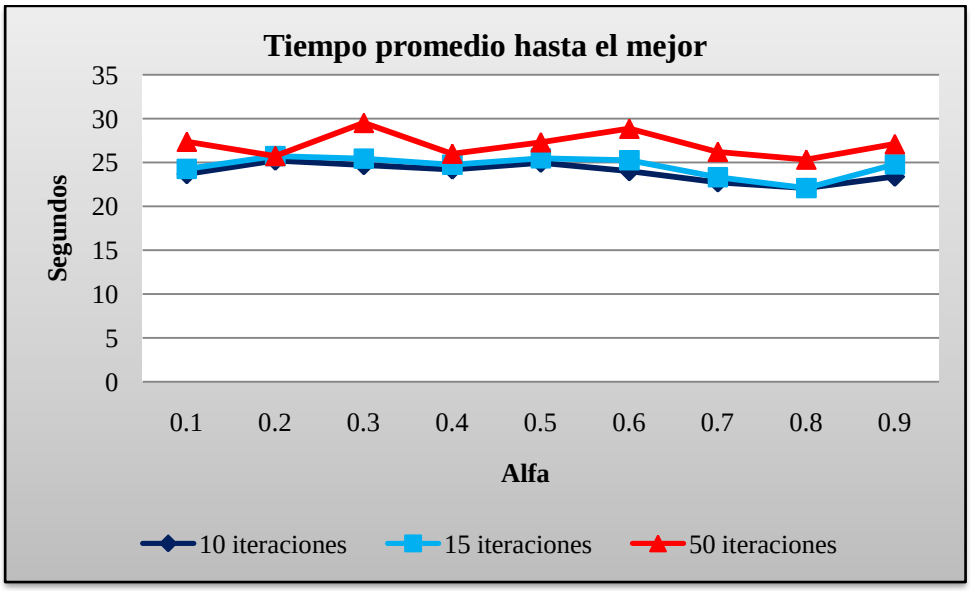

Figura 4.3. Gráfica comparativa para 10, 15 y 50 iteraciones de RT con E2.

4.5 Experimento 3: análisis del desempeño de la estrategia Vecindad Mejorada (VM) + Re-encadenamiento de Trayectorias (RT)

En este experimento se evalúa el desempeño del método que integra ambas propuestas de solución: la Vecindad mejorada (VM) y el Re-encadenamiento de trayectorias (RT) con sus 2 estrategias (E1 y E2).

4.5.1 Objetivo

Evaluar el desempeño de la propuesta de solución VM + RT con E1 y E2 con respecto a la solución de referencia reportada en [González 2004].

4.5.2 Procedimiento

Para este experimento se utilizaron diferentes valores de alfa (α). El número de iteraciones utilizado 10, 15 y 50 iteraciones. Posteriormente se hará un comparativo utilizando la solución de referencia reportada en [González 2004] y los resultados de este experimento.

4.5.3 Resultados

Las tablas 4.13, 4.14 y 4.15 muestran los resultados para 10, 15 y 50 iteraciones respectivamente para VM + RT E1 y las tablas 4.15, 4.16 y 4.17 muestran los resultados para 10, 15 y 50 iteraciones respectivamente utilizando VM + RT E2.

Tabla 4.13. Resultados acumulados para 10 iteraciones de la propuesta de solución VM + RT E1.

α (alfa)	Iteraciones promedio hasta encontrar el óptimo	Tiempo promedio hasta encontrar el óptimo	Tiempo promedio en procesar la instancia	Número de óptimos
0.1	10.07	24.67	52.90	86
0.2	10.5	25.66	53.98	86
0.3	10.27	25.03	53.68	85
0.4	9.9	24.43	53.54	88
0.5	10.73	25.47	53.16	86
0.6	9.97	24.28	52.56	87
0.7	9.5	22.71	52.13	84
0.8	9.27	22.36	52.84	86
0.9	9.6	23.12	52.88	84

Tabla 4.14. Resultados acumulados para 15 iteraciones de la propuesta de solución VM + RT E1.

α (alfa)	Iteraciones promedio hasta encontrar el óptimo	Tiempo promedio hasta encontrar el óptimo	Tiempo promedio en procesar la instancia	Número de óptimos
0.1	10.37	25.34	67.64	86
0.2	10.87	26.09	68.82	87
0.3	10.57	25.75	68.53	85
0.4	10.23	24.96	66.77	89
0.5	10.97	25.99	68.18	87
0.6	10.43	25.54	67.61	87
0.7	10.13	23.51	66.18	86
0.8	9.27	22.34	66.75	86
0.9	9.93	23.87	67.98	85

Tabla 4.15. Resultados acumulados para 50 iteraciones de la propuesta de solución VM + RT E1.

α (alfa)	Iteraciones promedio hasta encontrar el óptimo	Tiempo promedio hasta encontrar el óptimo	Tiempo promedio en procesar la instancia	Número de óptimos
0.1	11.9	28.32	116.76	89
0.2	10.87	26.06	114.67	87
0.3	11.33	31.55	119.33	89

0.4	11.33	27.11	116.26	89
0.5	12.47	28.09	118.35	89
0.6	11.9	28.64	118.28	90
0.7	11.8	26.63	115.63	88
0.8	11.07	25.89	116.77	89
0.9	12.27	28.34	118.16	88

4.5.4 Análisis de los resultados del experimento 3: VM + RT E1

En la Tabla 4.13 se observa que para 10 iteraciones de VM + RT E1, se logran 88 óptimos de los 90 posibles con un valor de α=0.4, respecto a la calidad de la solución. Al evaluar la eficiencia se toma en cuenta el tiempo promedio hasta el mejor, se observa que tiene un desempeño aceptable solamente superado por α=0.6, α=0.7, α=0.8 y α=0.9. La Tabla 4.14 muestra los resultados para 15 iteraciones de VM + RT E1. Evaluando la calidad se observa que para α=0.4 se logran 89 óptimos de los 90 posibles. Con respecto a la eficiencia se observa que su desempeño es superado por los valores de α=0.7, α=0.8 y α=0.9, ya que requiere de 24.96 segundos para encontrar el mejor.

En la Tabla 4.15 se muestran los resultados obtenidos para 50 iteraciones de VM + RT E1. Al evaluar la calidad de la solución se observa que para el valor de α=0.6 se logran los 90 óptimos posibles. Con respecto a la eficiencia de la solución, requiere de un mayor tiempo para encontrar el óptimo que el resto de los valores de α, solamente para α=0.3 el tiempo hasta encontrar el óptimo es mayor que para α=0.6. De esta manera se puede determinar que la mejor configuración para VM + RT E1 es con un valor de α=0.6 y 50 iteraciones.

En la Figura 4.4 se observa la gráfica comparativa del tiempo promedio hasta el mejor para 10, 15 y 50 iteraciones de la solución propuesta VM + RT E1. En la grafica se observa claramente que para 10 y 15 iteraciones la diferencia del tiempo requerido hasta el mejor es mínima sobre todo para α=0.8. En cambio para 50 iteraciones se aprecia que existe un incremento significativo, sobresaliendo el valor de α=0.3 y α=0.6.

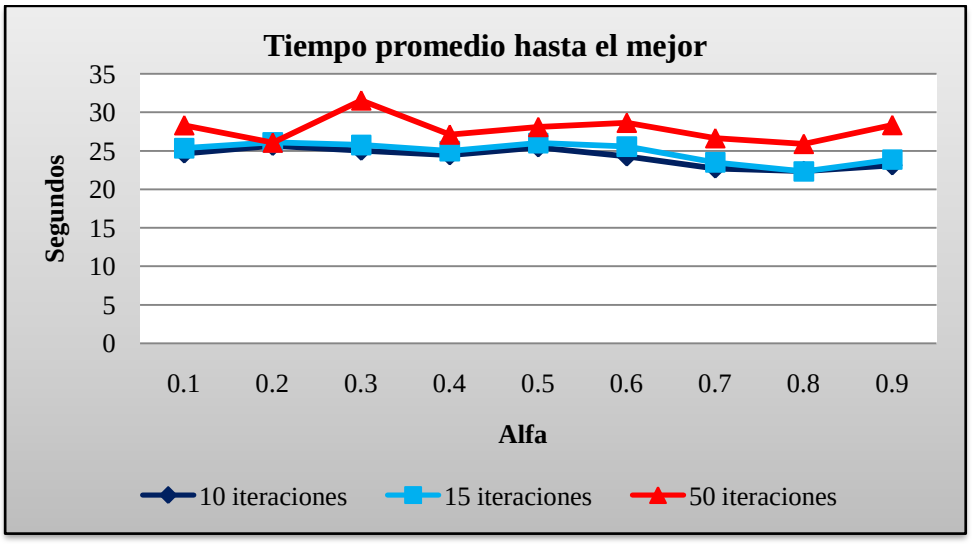

Figura 4.4. Gráfica comparativa para 10, 15 y 50 iteraciones de VM + RT con E1.

Tabla 4.16. Resultados acumulados para 10 iteraciones de la propuesta de solución VM + RT E2.

α (alfa)	Iteraciones promedio hasta encontrar el óptimo	Tiempo promedio hasta encontrar el óptimo	Tiempo promedio en procesar la instancia	Número de óptimos
0.1	9.73	23.46	52.49	86
0.2	10.23	24.45	53.09	86
0.3	10.17	24.59	52.98	86
0.4	9.87	23.97	53.06	88
0.5	10.47	24.72	52.29	86
0.6	9.87	23.79	52.17	88
0.7	9.63	22.41	51.62	85
0.8	9.17	22.05	52.32	86
0.9	9.63	22.92	52.19	83

Tabla 4.17. Resultados acumulados para 15 iteraciones de la propuesta de solución VM + RT E2.

α (alfa)	Iteraciones promedio hasta encontrar el óptimo	Tiempo promedio hasta encontrar el óptimo	Tiempo promedio en procesar la instancia	Número de óptimos
0.1	10.03	24.11	66.71	86
0.2	10.87	25.78	68.19	88
0.3	10.47	25.31	67.83	86
0.4	10.2	24.52	66.28	89
0.5	10.7	25.25	67.32	87
0.6	10.33	25.03	67.14	88
0.7	9.87	23.04	65.88	86
0.8	9.17	22.04	66.32	86
0.9	10.17	24.37	67.44	86

Tabla 4.18. Resultados acumulados para 50 iteraciones de la propuesta de solución VM + RT E2.

α (alfa)	Iteraciones promedio hasta encontrar el óptimo	Tiempo promedio hasta encontrar el óptimo	Tiempo promedio en procesar la instancia	Número de óptimos
0.1	11.57	27.05	114.63	89
0.2	10.87	25.76	116.24	88
0.3	12.93	29.38	118.31	89
0.4	11.3	26.69	115.96	89
0.5	12.2	27.34	117.91	89
0.6	11.5	27.92	117.76	90
0.7	11.53	26.17	115.69	88
0.8	10.97	25.57	115.84	89
0.9	11.7	26.90	117.01	88

4.5.5 Análisis de los resultados del experimento 3: VM + RT E2

La Tabla 4.16 muestra los resultados para 10 iteraciones de VM + RT E2. Evaluando con respecto a la calidad se puede apreciar que para los valores de α=0.4 y α=0.6 se obtiene el mayor número de óptimos al lograr 88 de los 90 posibles. Al evaluar con respecto a la eficiencia vemos que para α=0.6 le toma un menor tiempo hasta encontrar el óptimo con 23.79 segundos y para α=0.4 requiere de 23.97 segundos. En la Tabla 4.17 se observa que para 15 iteraciones de VM + RT E2 el mejor resultado se obtiene con un valor de α=0.4 al lograr 89 óptimos de los 90 posibles. En cuanto a la eficiencia, se aprecia que para α=0.4 requiere de 24.52 segundos hasta encontrar el óptimo, el rendimiento es solamente superado por α=0.1, α=0.7, α=0.8 y α=0.9.

La Tabla 4.18 muestra los resultados obtenidos para 50 iteraciones de VM + RT E2. En ella se observa que al evaluar la calidad de la solución con un valor de α=0.6 se logran los 90 óptimos. Con respecto a la eficiencia, se observa que el desempeño no es el mejor ya que requiere de 27.92 segundos hasta encontrar el óptimo y solamente para el valor de α=0.3 se necesita de más tiempo.

De esta manera podemos determinar que para VM + RT E2 el mejor desempeño se obtiene con 50 iteraciones y un valor de α=0.6, logrando los 90 óptimos posibles.

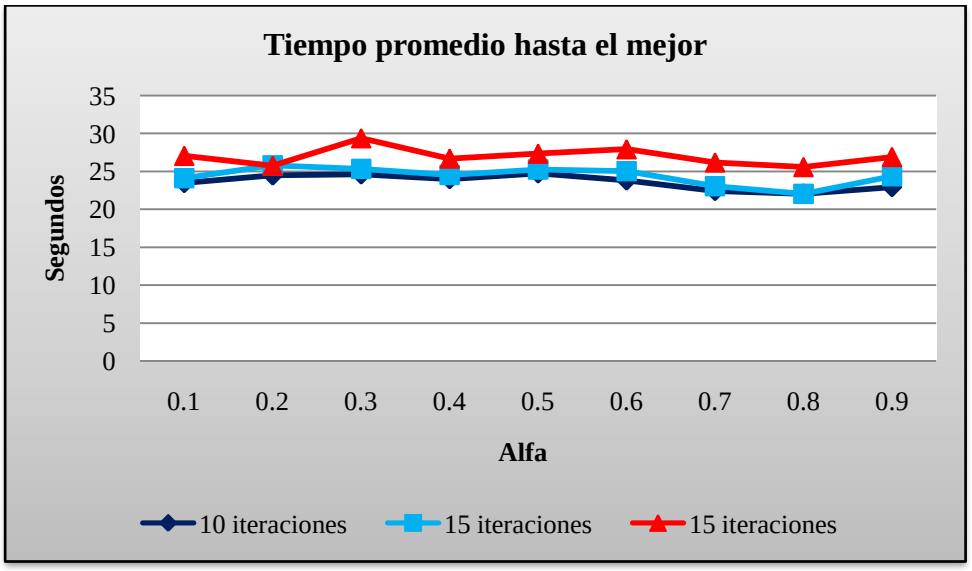

Figura 4.5. Gráfica comparativa para 10, 15 y 50 iteraciones de VM + RT con E2.

La Figura 4.5 muestra la gráfica comparativa del tiempo promedio hasta el mejor para 10, 15 y 50 iteraciones de la solución propuesta VM + RT E2. En ella podemos observar que para 10 y 15 iteraciones la diferencia es mínima, solamente en el caso de α=0.2 y α=0.6 los valores para 15 iteraciones se incrementan un poco. Para 50 iteraciones los valores se incrementan considerablemente, esto debido a que al realizar un mayor número de iteraciones requiere de más tiempo.

4.6 Comparativo de las propuestas de solución

Una vez que se ha experimentado con cada una de las soluciones propuestas y obtenido los resultados se pudo determinar la mejor configuración para cada una de ellas en cuanto al valor de alfa (α) y el número de iteraciones. La Tabla 4.19 muestra los resultados para cada una de las soluciones propuestas de acuerdo a la mejor configuración obtenida. En ella podemos observar que para lograr los 90 óptimos posibles se requiere de 50 iteraciones en cada una de las soluciones.

Tabla 4.19. Tabla comparativa de las propuestas de solución.

Solución	Iter	α (alfa)	Iteraciones promedio hasta encontrar el óptimo	Tiempo promedio hasta encontrar el óptimo	Tiempo promedio en procesar la instancia	Número de óptimos
VM	50	0.6	11.5	28.02	118.05	90
RT E1	50	0.4	11.63	28.65	134.30	90
RT E2	50	0.6	11.9	28.85	133.31	90
VM + RT E1	50	0.6	11.9	28.64	118.28	90
VM + RT E2	50	0.6	11.5	27.92	117.76	90

4.6.1 Análisis del comparativo

Como se observa en la Tabla 4.19, evaluando con respecto a la calidad, todas las soluciones propuestas obtienen los 90 óptimos. Al evaluar la eficiencia tomando en cuenta el tiempo promedio hasta el mejor, la propuesta que mejor desempeño tiene al requerir de un menor tiempo es VM +RT E2 con 27.92 segundos.

La Figura 4.6 muestra la gráfica comparativa del tiempo promedio hasta el mejor de cada una de las mejores configuraciones en cuanto al valor de alfa (α) y al número de iteraciones de las propuestas de solución. Como se puede observar en la gráfica, las soluciones VM y VM + RT E2 son las que menor tiempo requieren para lograr los 90 óptimos; a diferencia de las soluciones RT E1, RT E2 Y VM + RT E1 cuyos tiempos hasta encontrar el óptimo son mayores. De aquí podemos observar claramente que la mejor solución de acuerdo a la calidad y eficiencia es VM + RT E2 con 50 iteraciones y un valor de α=0.6.

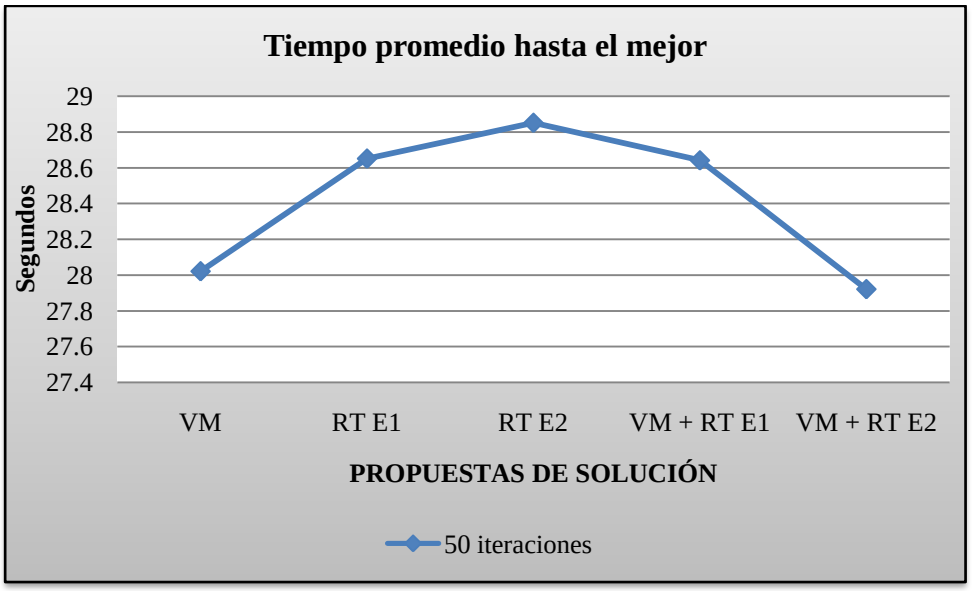

Figura 4.6. Gráfica comparativa para las propuestas de solución con 50 iteraciones.

4.7 Resultados de la solución de referencia

La Tabla 4.20 muestra los resultados obtenidos para 10, 15 y 50 iteraciones de la solución de referencia reportada en [González 2004]. Cabe mencionar que la solución de referencia no utiliza un alfa (α).

Tabla 4.20. Resultados acumulados de la solución de referencia.

Iter	Iteraciones promedio hasta encontrar el óptimo	Tiempo promedio hasta encontrar el óptimo	Tiempo promedio en procesar la instancia	Número de óptimos
10	12.3	29.98	58.85	85
15	12.83	30.92	76.68	87
50	13.3	31.61	139.35	88

Al evaluar con respecto a la calidad de la solución, la Tabla 4.20 muestra que el mejor desempeño se obtiene con 50 iteraciones al lograr 88 de los 90 óptimos posibles, ya que para 10 y 15 iteraciones se logran 85 y 87 óptimos respectivamente. Con respecto a la eficiencia el mejor desempeño se obtiene con 10 iteraciones al requerir solamente 29.98 segundos hasta encontrar el óptimo. Para 50 iteraciones requiere de 31.61 segundos, siendo este el más pobre

desempeño. Ya que el principal indicador para medir el desempeño de la solución es la calidad, se determina que el mejor resultado de la solución de referencia reportado en [González 2004] se obtiene con 50 iteraciones.

4.8 Comparativo de la mejor solución propuesta y la solución de referencia

Una vez que se ha determinado que la mejor solución propuesta es VM + RT E2 con 50 iteraciones y un valor de α=0.6, y que para la solución de referencia reportada en [González 2004] se obtiene con 50 iteraciones se hace un comparativo entre ambas. La Tabla 4.21 muestra dicho comparativo.

Tabla 4.21. Tabla comparativa de la mejor solución propuesta y la solución de referencia.

Solución	Iter	Alfa α	Iteraciones promedio hasta encontrar el óptimo	Tiempo promedio hasta encontrar el óptimo	Tiempo promedio en procesar la instancia	Número de óptimos
[González 2004]	**50**	**----**	**13.3**	**31.61**	**139.35**	**88**
VM + RT E2	50	0.6	11.5	27.92	117.76	90

Como se observa en la Tabla 4.21, al evaluar con respecto a la calidad de la solución se aprecia que la solución propuesta VM + RT E2 logra los 90 óptimos posibles y la solución de referencia [González 2004] obtiene solamente 88 óptimos de los 90 posibles. Así, en cuanto a la calidad, se observa que VM + RT E2 tiene un mejor desempeño. Evaluando con respecto a la eficiencia, se observa que la solución propuesta VM + RT E2 requiere de 27.92 segundos hasta encontrar el óptimo mientras que la solución de referencia de [González 2004] necesita de 31.61. De esta manera se observa que en cuanto a la eficiencia la solución propuesta VM + RT E2 tiene un mejor desempeño. Así, se puede determinar que la solución propuesta VM + RT E2 tiene un mejor desempeño que la solución de referencia reportada en [González 2004].

Capítulo 5

CONCLUSIONES Y TRABAJOS FUTUROS

Este capítulo presenta las aportaciones de esta investigación a la solución del problema ROCIS de acuerdo a las propuestas de solución mostradas en el capítulo 3. De igual manera, se describen los posibles trabajos futuros identificados durante su desarrollo.

5.1 Conclusiones

A continuación se presentan las aportaciones más importantes de este trabajo.

a) Se aporta una mejora en la segunda etapa (*generación de vecinos*) de la estructura de la solución tabú reportada en [González 2004]. Dicha aportación consiste en mejorar específicamente el cálculo de los *costos relativos* (r_i) de cada proveedor (ver sección 3.2.3).

   Los *costos relativos* de cada proveedor se utilizan para formar las listas de candidatos de *inserción*, *eliminación* e *intercambio* de los proveedores. La aportación consiste en considerar la capacidad (b_i) del proveedor *i* al momento de calcular su *costo relativo*. Con esta aportación se diversifica el proceso de solución, es decir; se puede elegir entre un mayor número de posibles soluciones logrando con esto que es espacio explorado crezca considerablemente. Esto permite obtener la mejor solución en un menor tiempo posible y con un menor esfuerzo.

b) Se aporta una mejora en la tercera etapa (*búsqueda local*) de la estructura de la solución tabú reportada en [González 2004]. Esta aportación consiste en realizar un re-encadenamiento de trayectorias como se describe en la sección 3.3, con sus 2 estrategias (ver secciones 3.3.1.1 y 3.3.1.2).

   Esta aportación consiste en intensificar la búsqueda de la solución en un menor número de iteraciones, por lo cual el tiempo necesario para lograr el óptimo se reduce considerablemente.

Las dos aportaciones más importantes que se realizaron durante el desarrollo de este trabajo se enfocan en la diversificación e intensificación del proceso de búsqueda. Una vez que se logaron los 90 óptimos posibles, se trabajó en reducir el tiempo de búsqueda el uso de re-encadenamiento de trayectorias. Logrado este propósito, se puede determinar que dichas aportaciones son de gran importancia ya que además de lograr el mayor número de óptimos posibles se realiza una búsqueda de la solución de una manera mucho más intensa cuando se combinan las dos aportaciones y trabajan de manera conjunta (ver sección 3.4).

5.2 Trabajos futuros

Durante el desarrollo de esta investigación se han identificado claramente las siguientes líneas de investigación que pueden contribuir de manera significativa a mejorar la solución al problema ROCIS en cuanto a eficiencia y calidad.

a) Clasificar aquellas instancias de prueba que no pueden ser resultas de manera óptima con la solución actual. Una forma de hacerlo es detectar las instancias más difíciles y que no pueden ser resueltas y hacer un análisis de las mismas para identificar si existen características similares entre ellas que las hacen difíciles de resolver de manera óptima.

   Hasta el momento se ha trabajo con el grupo de instancias de prueba reportada en [González 2004] con un número máximo de 10 plantas y 20 proveedores, pero una alternativa es generar un grupo de instancias de mayor tamaño como lo reporta [González 2006] con 10 plantas y 40 proveedores. Al ser de mayor tamaño las instancias se hacen mucho más difíciles de resolver

b) Implementar un mecanismo de diversificación en la búsqueda de la solución. Dado que conforme las instancias sean de un mayor tamaño (número de plantas y proveedores), el espacio de solución crece significativamente. Por ejemplo, para 20 proveedores el número total de posibles soluciones es $2^{20} = 1048576$, las listas de candidatos contienen el siguiente número de posibles soluciones: inserción 3, eliminación 3 e intercambio $\frac{m^2 - m}{8}$ siendo *m* el número de proveedores que en este caso es 20, por lo tanto se tienen 47 posibles soluciones; dando un total de 53 soluciones a analizar en cada iteración. Para instancias pequeñas el espacio de búsqueda se explora de manera satisfactoria. Cuando se trata de instancias grandes como en el caso de 40 proveedores, el número total de posibles combinaciones es de $2^{40} = 1.099511628^{12}$ y el número de

posibles soluciones a analizar en cada iteración es de 201; con lo cual es fácil apreciar que el espacio a explorar es extremadamente pequeño en comparación con el espacio de soluciones.

c) Otra línea de investigación es diseñar un mecanismo de diversificación para la búsqueda utilizando una memoria de largo plazo que pueda ayudar a que el espacio de soluciones sea explorado más ampliamente, llevando un contador de cuantas veces ha sido seleccionado un proveedor para que de esta manera se alcance un mayor número de soluciones.

d) Finalmente se plantea otra investigación enfocada a utilizar las estrategias descritas en [Glover 1997] de la búsqueda tabú en cuanto a la fase de diversificación, ya que la solución presentada en este trabajo implementa algunas de las técnicas sugeridas en dicha referencia; pero se puede experimentar con aquellas que aún no se han considerado en esta tesis.

Referencias

[Mulvey 1995] Mulvey J.M., R.J. Vanderbei and S.A. Zenios 1995. Robust optimization of large-scale systems. Operations Research 43(2), p.264-281.

[Laguna 1999] Laguna, M., R. Martí and V. Campos (1999) "Intensification and iversification with Elite Tabu Search Solutions for the Linear Ordering Problem," Computers and Operations Research, vol. 26, pp. 1217-1230.

[LINDO API 2002] LINDO Systems Inc. Chicago Illinois, "LINDO API User's Manual".

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

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

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

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

[Hodder 1982] Hodder, J.E. and J.V. Jucker (1982) "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, pp. 248-258.

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

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

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

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

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

[Escudero 1993] Escudero, L.F., P.V. Kamesam, A.J. King y R.J.-B. Wets (1993), "Production

Planning via Scenario Modelling", Annals of Operations Research, vol. 43, pp. 311-335.

[González 2004] Jose Luis Gonzalez Velarde, "A Benders-based heuristic for the robust capacitated international sourcing problem.", IIE Transactions, November 1, 2004, Volume: 36 Issue: 11 Page: 1125(9), Institute of Industrial Engineers, Inc. (IIE).

[González 2006] González-Velarde J. L. y Martí R. "Adaptive Memory Programming for the Robust Capacitated International Sourcing Problem", 2006.

[Gómez 2007] Santiago Gómez Carpizo, "Búsqueda Tabú aplicada al problema Robusto de Abastecimiento Internacional con Capacidad Finita (ROCIS)", Tesis de Maestría, Instituto Tecnológico de Cd. Madero (en proceso).

[Ignizio 1994] James P. Ignizio and Tom M. Cavalier, "Linear programming". Prentice Hall, 1994.

[Loomba 1964] Loomba, N.P. Linear Programming: An introductory analysis. McGraw-Hill, New York, 1964

[Glover 1997] Fred Glover and Manuel Laguna, "Tabu Search". Kluwer Academic Publishers, 1997.