CONTENIDO

FIGURAS

TABLAS

3.1 Descripción general

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

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

Figura 3.1 Algoritmo de la solución referencia

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

Figura 3.1 Continuación …

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

Figura 3.1 Continuación…

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

Figura 3.1 Continuación …

3.2 Representación binaria y decimal de las soluciones

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

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

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

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

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

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

3.3 Estructuras de datos utilizadas

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

Tabla 3.1 Estructuras utilizadas por la interfaz de LINDO

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

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

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

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

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

Tabla 3.3 Continuación …

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

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

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

Tabla 3.5 Estructura inputdata

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

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

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

Tabla 3.6 Continuación …

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

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

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

Tabla 3.7 Continuación …

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

3.4 Funciones utilizadas

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

Tabla 3.8 Funciones utilizadas para resolver el problema ROCIS.

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

3.4.1 GeneraciónVoraz()

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

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

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

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

3.4.1.1 Función Initialize()

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

Algoritmo: Initialize

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

Entrada: Datos de la instancia a evaluar

Salida: solución _ actual

Figura 3.2 Algoritmo de la función Initialize()

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

Figura 3.2 Continuación …

3.4.1.2 Función select_min()

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

Función select_min()

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

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

Salida: Selección del proveedor que cumpla con dicho objetivo.

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

Figura 3.3 Algoritmo de la función select_min()

3.4.2 Función BúsquedaLocal()

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

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

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

Tabla 3.11 Funciones utilizadas en la función BúsquedaLocal()

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

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

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

Algoritmo: Búsqueda Local

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

Entrada: Datos de la instancia a evaluar y la solución_inicial

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

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

Figura 3.4 Algoritmo de la función BusquedaLocal ()

3.4.2.1 Función form_list()

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

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

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

Tabla 3.13 Listas tabú de inserción, eliminación e intercambio.

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

Tabla 3.13 Continuación.

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

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

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

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

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

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

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

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

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

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

Algoritmo: form_list

Objetivo: Generar una vecindad a partir de una solución inicial

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

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

1 Para i =1 $\rightarrow$ Num_Proveedores

2 Para s =1 $\rightarrow$ Num_Escenarios

3 $$E(\pi_i) = \sum_{s \in S} p_s \pi_{is}$$

4 Fín Para

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

6 Fín Para

Figura 3.5 Algoritmo de la función form_list ()

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

Figura 3.5 Continuación …

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

Figura 3.5 Continuación …

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

Figura 3.5 Continuación …

3.4.2.2 Función select_move()

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

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

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

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

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

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

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

Algoritmo: *select_move*

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

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

Salida: Selección de la mejor solución del conjunto elite

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

Figura 3.6 Algoritmo *select_move*

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

Figura 3.6 Continuación …

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

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

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

Figura 3.6 Algoritmo *select_move*

3.4.2.3 Función execute_move

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

Algoritmo: execute_move

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

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

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

Figura 3.7 Algoritmo execute_move

```
1    Si Operador = Inserción
2        Solución_inicial[indice_proveedor]=1
3        Capacidad_conjunta=
         Capacidad_conjunta + Capacidad[indice_proveedor]
4        Del_Tabu[indice_proveedor] = duracion_tabu
5    Pero Si Operador = Eliminación
6        Solución_inicial[indice_proveedor]=0
7        Capacidad_conjunta=
         Capacidad_conjunta - Capacidad[indice_proveedor]
8
     Ins_Tabu[indice_proveedor] = duracion_tabu
9    Pero Si Operador = Intercambio
10       Solución_inicial[indice_proveedor_ins]=1
11       Solución_inicial[indice_proveedor_del]=0
         Capacidad_conjunta= Capacidad_conjunta +
12           Capacidad[indice_proveedor_ins] -
             Capacidad[indice_proveedor_del]

13
     Swap_Tabu[indice_proveedor_ins][indice_proveedor_del] = duracion_tabu
14   End Si
15   Si  mejor_solucion_encontrada < mejor_solucion_global
16           mejor_solucion_global ← mejor_solucion_encontrada
17   End Si
```

Figura 3.7 Continuación …