CAPÍTULO 4

DISEÑO FUNCIONAL

Para la resolución del problema *VSP* operacional, se ha creado una herramienta basada en el procedimiento de resolución descrito en el capítulo anterior. Ésta ha sido desarrollada en el entorno de programación *Visual Basic 6.0*, pues ya se contaba con otra similar desarrollada por el Departamento.

Visual Basic 6.0 posee un ambiente de desarrollo completamente gráfico que facilita bastante la creación de interfaces así como la programación en sí. *Visual Basic* (*Visual Studio*) constituye un entorno de desarrollo integrado (*Integrated Development Enviroment*, *IDE*) que ha sido empaquetado como un programa de aplicación, es decir, consiste en un editor de código, un depurador, un compilador y un constructor de interfaz gráfica o *GUI*.

Los proyectos en *Visual Basic* pueden estar compuestos por varios tipos de archivos (formularios, módulos, controles de usuario, etc), pero en este proyecto sólo se han usado módulos de datos. Éstos son archivos que contienen las funciones que nosotros creamos. Lo correcto es tener un módulo principal donde aparezcan las llamadas a las funciones del resto de módulos. Los elementos del proyecto *VIRUS.vbp* se encuentran en la figura 4.1.

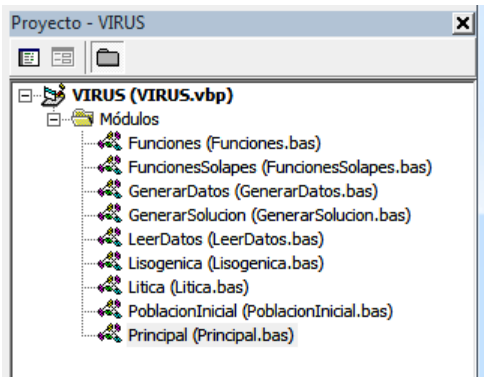

Figura 4.1. *Módulos del proyecto*

4.1. PROCEDIMIENTO PRINCIPAL

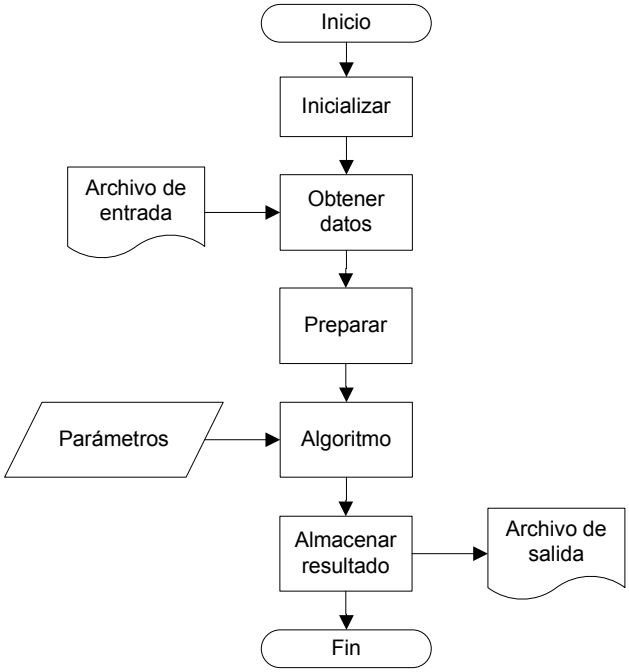

Figura 4.2. *Main*

En la figura se muestra un esquema del funcionamiento de la función *main* que está en *Principal.bas*. En este módulo es dónde están definidos todos los tipos de estructuras utilizadas, además de las variables globales.

Lo primero que se hace es inicializar los valores de las variables que contienen la ruta de entrada, para leer ficheros, y la ruta de salida, para escribir en la base datos los resultados obtenidos. A continuación se obtienen los datos del fichero de entrada que se le haya indicado. En este fichero vienen los datos del número de trabajos a realizar, de cada trabajo cuáles son su inicio y fin de la ventana de comienzo, su duración y su peso, también viene el número de máquinas a usar y el coste de uso (este dato no importa, ya que no se consideran clases de máquinas, y el coste de procesar en una máquina o en otra es el mismo).

La función *ObtenerDatos*, está en el módulo *LeerDatos.bas*. En esta función se almacenan los datos del fichero en una serie de variables, y las características de cada trabajo se guardan en un vector de estructuras, donde cada estructura es un trabajo, y sus campos son esos valores.

Lo siguiente es preparar esos datos de entrada para que sea más cómoda su posterior utilización. En la función *Preparar*, que está en el módulo *Principal.bas*, se calcula cuál es el horizonte temporal del problema, después se ordenan los trabajos de menor a mayor instante de inicio de la ventana de comienzos (lo cual se hace con *OrdenarTrabajos* del módulo *Funciones.bas*; esta función ha sido proporcionada por el tutor del proyecto). Con los trabajos ordenados de esta forma, se crean una serie de estructuras (una por cada trabajo), en las cuáles se indican los solapamientos que se producen entre los trabajos, esto es cuando la ejecución de un trabajo *j* que empieza en un instante *k* de su ventana de comienzos coincide en algún momento con otro trabajo. Esto se hace con las funciones *ObtenerSolapes* y *OrdenarSolapes* (esta última ordena dentro de una estructura los trabajos que solapan con *j* de mayor a menor peso) del módulo *FuncionesSolapes.bas*.

Una vez están los datos del problema preparados para su uso, se pasa a la ejecución del algoritmo vírico introducido en el capítulo anterior, al cual hay que indicarle cuáles son los valores del número de iteraciones, el tamaño de la población, las probabilidades de que un elemento resulte de tipo *lítico* (p_{Liti}) o *lisogénico* (p_{Liso}), y el límite del número de réplicas (*LNR*) para el caso *lítico*, y el del número de iteraciones (*LIT*) para el caso *lisogénico*.

Cuando se acaba de ejecutar el algoritmo, se almacenan los resultados en una base de datos. Esta función (*AlmacenarResultado*) se encuentra en *Principal.bas*, y también ha sido proporcionada por el tutor del proyecto.

4.2. ALGORITMO

Este procedimiento, como se ve en la figura 4.3, comienza con la formación de una población inicial. Los elementos de ésta son posibles soluciones al problema, que se generan aleatoriamente, pero que no suelen ser óptimas.

Para la construcción de esta población se utiliza la función *ObtenerPoblacionInicial* del módulo *PoblacionInicial.bas*, que además, calcula qué

solución de las iniciales es la que tiene el mayor valor de la suma de pesos, y se almacena como la mejor solución inicial.

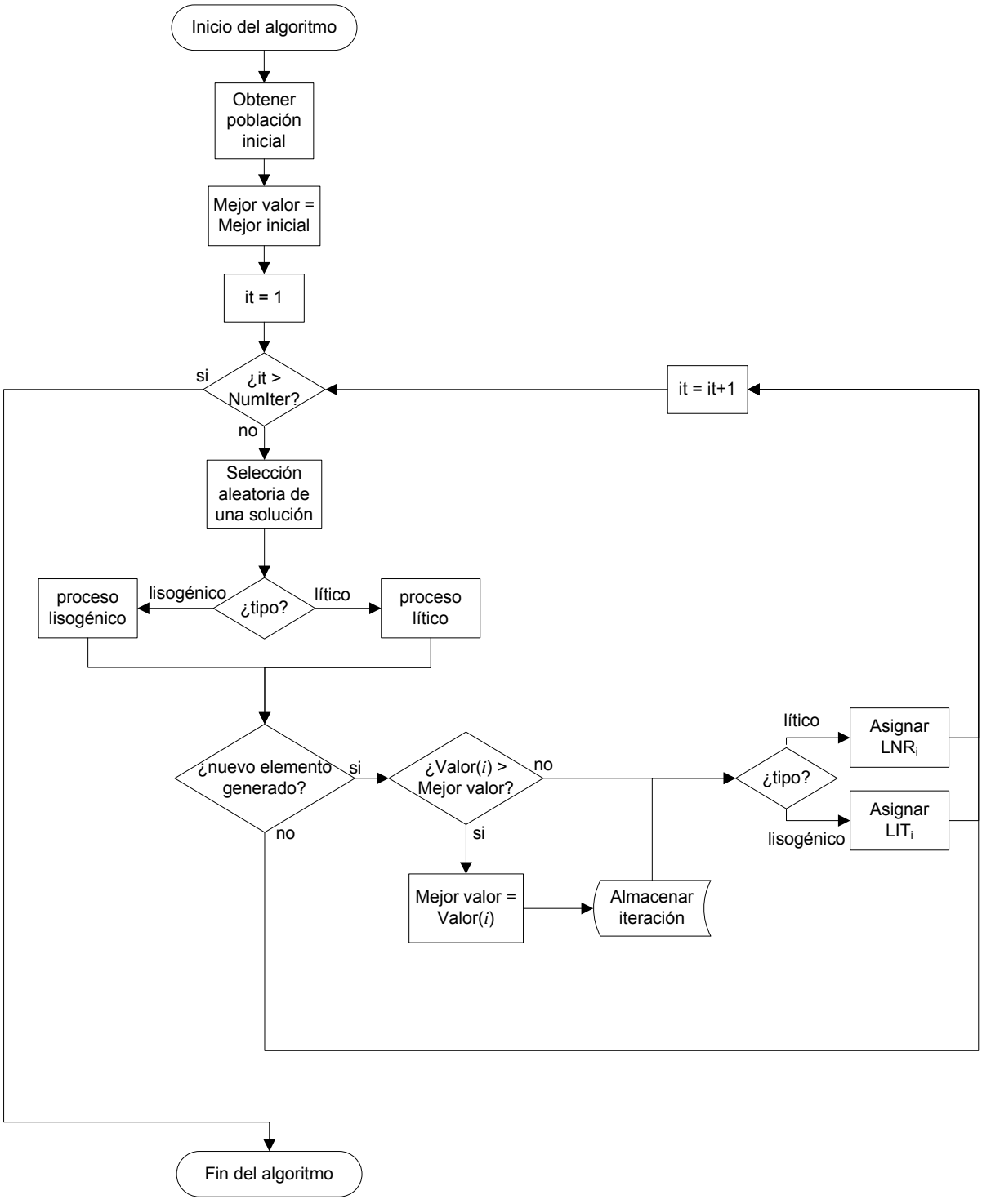

Figura 4.3. *Algoritmo*

Una vez obtenida la población inicial, se le define el mejor valor inicial como el mejor encontrado hasta el momento, y se pasa a la ejecución de un bucle, que se ejecutará hasta que se alcance el número de iteraciones definido.

En cada iteración del bucle se escoge aleatoriamente un elemento de la población, y según el tipo de éste (*lítico* o *lisogénico*), se le aplicará un tratamiento u otro.

Al acabar el procedimiento que le corresponda se comprueba si se ha obtenido una solución mejor de la obtenida hasta ese momento, y si es así, se almacena su valor, y la iteración en la que se ha producido esto.

Para finalizar el algoritmo se comprueba si en el procedimiento que se haya ejecutado se ha generado un nuevo elemento (se infectó una nueva célula), y si ese es el caso, dependiendo del tipo que sea, se le asigna un nuevo LNR_i o LIT_i.

4.2.1. SOLUCIÓN INICIAL

En esta función se recorren todas las células que forman parte de la población y se infectan, cada una de forma aleatoria. Este conjunto de células es un vector de estructuras, donde cada estructura es una posible solución al problema.

Para esto, como se ve en la figura 4.4, se recorre elemento a elemento de la población y se le da un genoma (codificación de la solución) y un tipo aleatorios.

Ya están infectadas las células, a continuación se calcula cuál de ellas posee el mejor valor de la solución, y ésta se almacena como la mejor inicial. Y para finalizar, se establecen los valores LNR_i y LIT_i según sea el caso, y las variables NR_i e IT_i se inicializan a cero en todas las células.

4.2.2. PROCESO LÍTICO

Éste es el procedimiento que se ejecuta cuando el elemento elegido para realizar la replicación (o la infección si está preparado) es de tipo *lítico.*

Como se ve e la figura 4.5, al comenzar lo primero que se hace es comprobar si el número de replicas que hay dentro de la célula son suficientes para realizar la propagación, y si este número es inferior al límite establecido, entonces se realiza una replicación del virus en el interior de la célula, y finaliza el proceso.

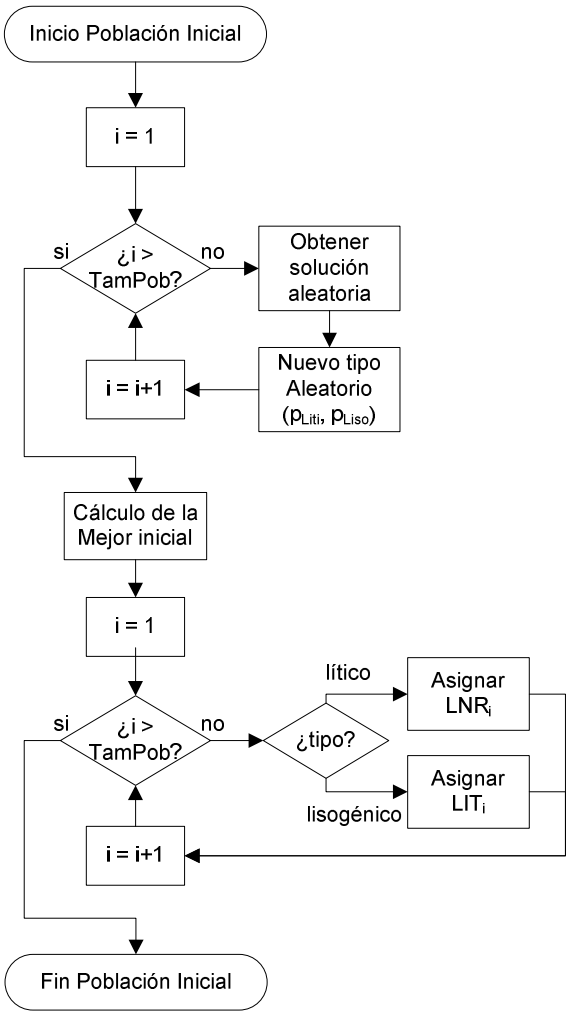

Figura 4.4. *ObtenerPoblacionInicial*

Si por el contrario, el número de replicas sí que ha llegado al límite ($NR_i \geq LNR_i$), entonces el programa pasa a realizar el cálculo de las posibles soluciones vecinas. Para este cálculo se recorren todos los trabajos uno a uno, y se va comprobando si cada trabajo *j* está o no presente en la solución; si no lo está, la vecina será la generada al introducir *j* en ésta como se ve en la figura 4.6 (vecindad por inserción), y en caso de que no pueda introducirse, entonces ese trabajo no genera vecina.

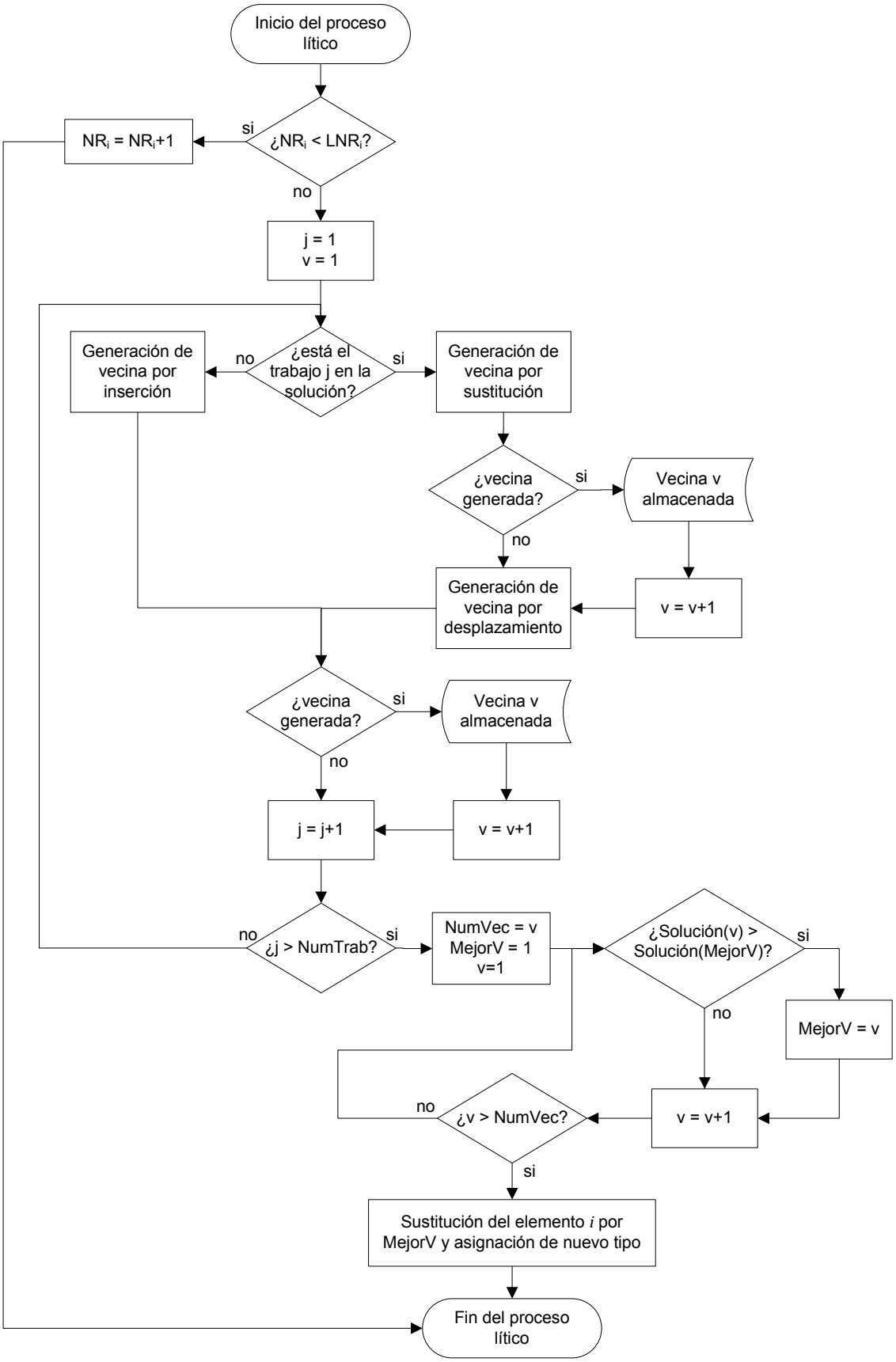

Figura 4.5. *Proceso lítico*

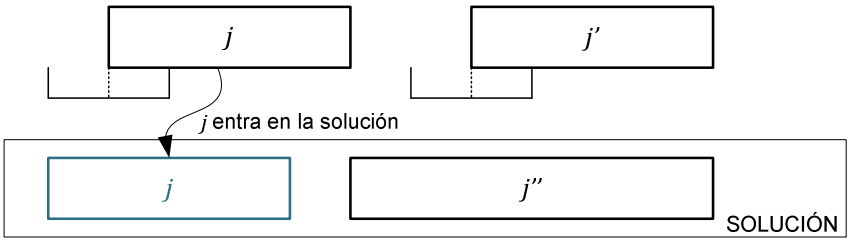

Figura 4.6. *Vecindad por inserción*

En el caso en que j sí esté en la solución se pueden generar dos posibles vecinas con ese trabajo. La primera es la que se forma al quitar j e introducir en su lugar otro trabajo que no esté en la solución (vecindad por sustitución, figura 4.7), si no se inserta ningún trabajo, la vecina se da por válida, ya que hay que permitir que la solución empeore. Una vez generada esta, la siguiente posible vecina es la que viene dada al desplazar el inicio de j por su ventana de comienzos (vecindad por desplazamiento, figura 4.8); si al desplazar j es posible introducir otro trabajo en la solución se forma una nueva vecina. Este proceso equivale, en términos del procedimiento, a la obtención de la $entrada_i$.

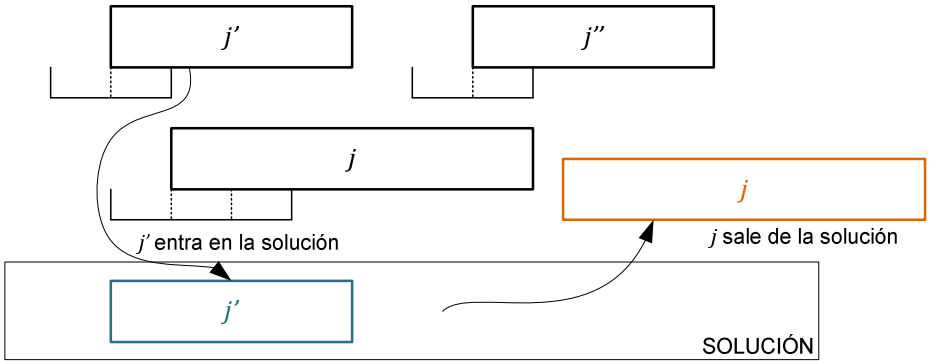

Figura 4.7. *Vecindad por sustitución*

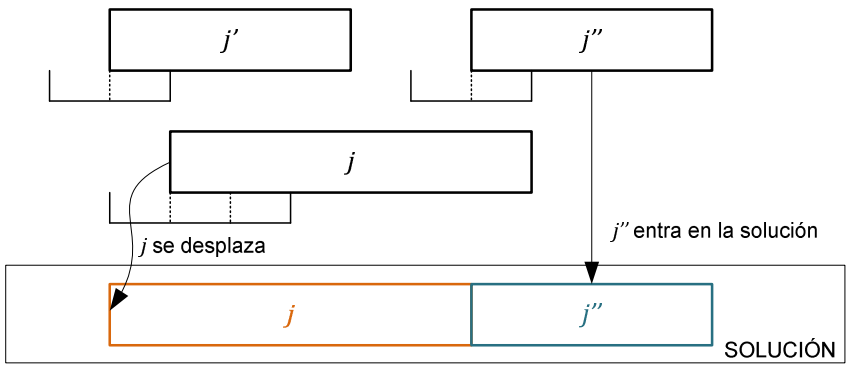

Figura 4.8. *Vecindad por desplazamiento*

Cuando la generación de vecindades termina, se comprueba cuál de ellas es la que ofrece mayor valor de la solución, siendo ésa la elegida para ser infectada, es decir, la que sustituirá a la actual. Hecho esto, el procedimiento *lítico* finaliza asignándole aleatoriamente a la nueva célula infectada un tipo de replicación con probabilidades p_{Liti} y p_{Liso}.

4.2.3. PROCESO LISOGÉNICO

Al igual que en el *lítico*, lo primero que se hace, tal y como se muestra en la figura 4.9, es comprobar si se va a intentar realizar la propagación, y para ello se chequea el número de iteraciones que ha sufrido esa célula (IT_i); si ese número es igual o superior que el límite establecido para ésta ($IT_i \geq LIT_i$), entonces se intenta, si no, simplemente se aumenta el número de iteraciones en una unidad y se finaliza el proceso.

Si el número de iteraciones resulta haber alcanzado el límite, lo que se hace es escoger aleatoriamente un trabajo, y si este está o no en la solución se actuará de una manera u otra.

En el caso en que el trabajo escogido (j) no esté en la solución, se intenta generar una vecina de dicha solución insertando un trabajo que no esté en ella (figura 4.6). Si no se consigue insertar, no se habrá generado la vecindad.

Si ese trabajo sí está en la solución, lo que se hará será intentar generar una vecindad, pero en este caso sustituyendo *j* por otro trabajo que no esté en la solución (figura 4.7). Y al igual que antes, si quitando el trabajo *j* de la solución no se consigue insertar ningún otro, entonces también se genera la vecindad para permitir que la solución empeore.

Por un camino u otro, si se genera la vecina, se sustituye la célula elegida por ella, y se le asigna un nuevo tipo de forma aleatoria. Y si no se genera, se acaba el proceso sin hacer nada más.

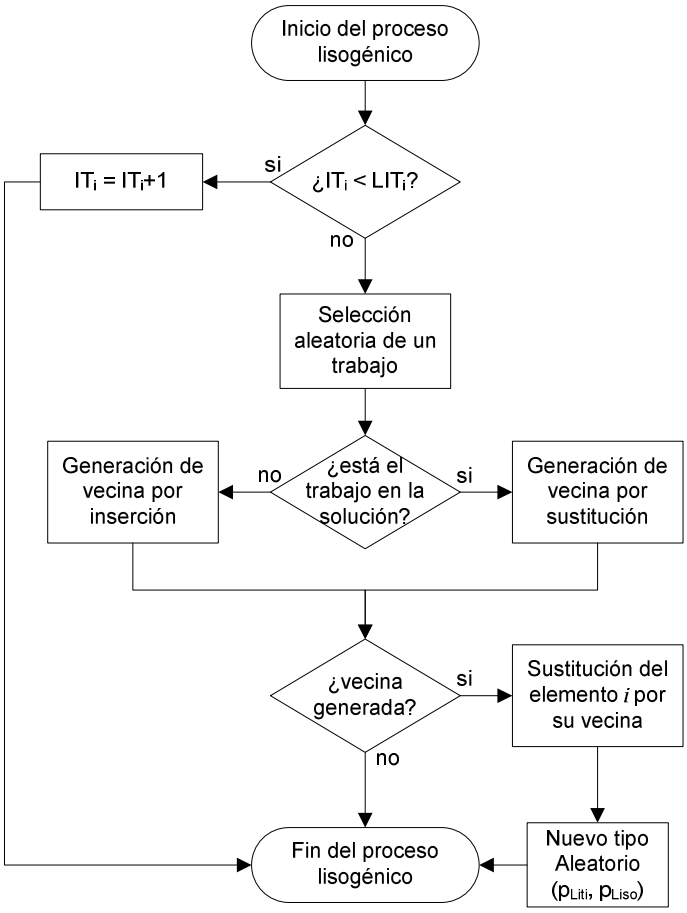

Figura 4.9. *Proceso lisogénico*

4.3. ILUSTRACIÓN

Para enseñar el funcionamiento del algoritmo, se va a mostrar cómo serían dos iteraciones del mismo para un problema pequeño de 10 trabajos.

Este ejemplo se muestra en la figura 4.10, donde vemos los momentos de llegada y salida de los trabajos, su duración, y entre paréntesis el peso de los mismos.

Con el procedimiento empezado, tomamos una iteración cualquiera *IT*, en la que nos encontramos a la población tal y como se ve en la figura 4.11. En la primera columna se muestran los trabajos escogidos y entre paréntesis la máquina donde se realiza ese trabajo (*m*) y el instante en el que comienza (*k*). En la segunda y tercera columnas vemos el valor, cuando proceda, del número de replicaciones (tipo *lítico*) o de iteraciones (tipo *lisogénico*), en las columnas 5 y 6 aparecen los valores límite

para ese número de replicaciones o iteraciones y, por último, en la columna 4 se muestra el valor de la solución sumando los pesos de los trabajos procesados.

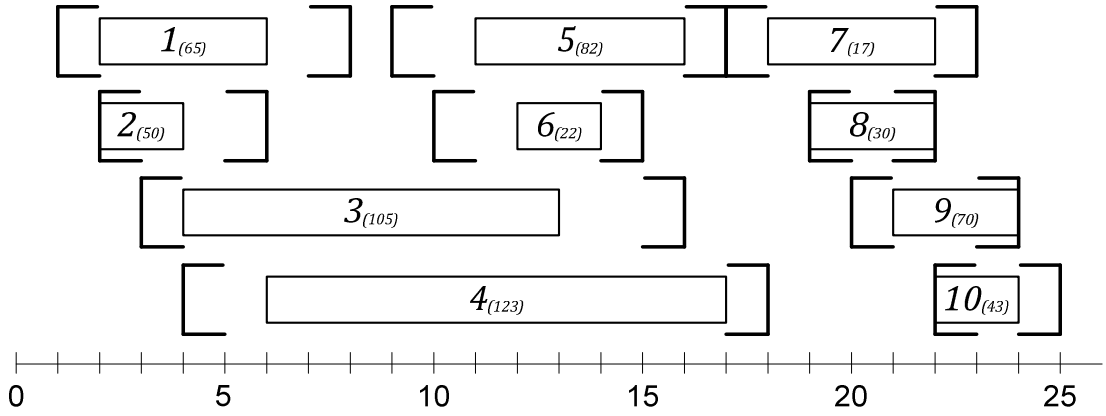

Figura 4.10. *Ejemplo de 10 trabajos*

Trabajos (m,k)	NR	IT	Valor	LNR	LIT
1(1,2) . 4(2,5) . 7(1,17)	0	IT_1	205	0	LIT_1
2(1,3) . 5(1,11) . 6(2,12) . 9(2,20)	NR_2	0	224	LNR_2	0
4(1,6) . 5(2,10) . 7(1,18) . 10(2,23)	0	LIT_3	265	0	LIT_3
3(1,5) . 4(2,4) . 8(1,19)	LNR_4	0	258	LNR_4	0
1(1,2) . 3(1,6) . 6(2,13) . 7(1,17)	NR_5	0	209	LNR_5	0

Figura 4.11. *Población en la iteración IT*

El algoritmo (figura 4.3) escoge aleatoriamente un elemento para procesarlo; suponemos que en esta primera iteración se escoge el elemento 4, que es de tipo *lítico*. Una vez escogido el elemento de la población y visto su tipo, se pasa a ejecutar el proceso del tipo indicado, siempre y cuando su NR_i o IT_i haya alcanzado el límite.

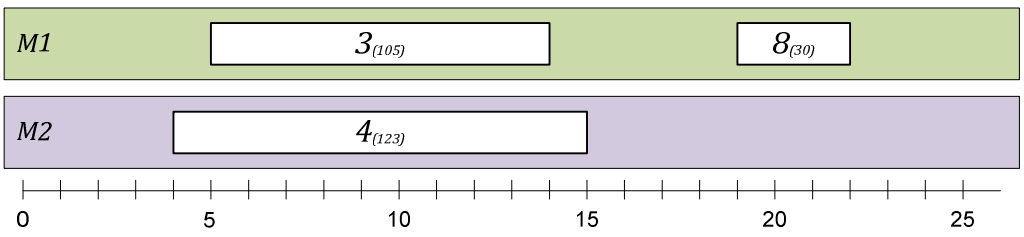

Figura 4.12. *Elemento 4 de la población*

En el proceso *lítico* se calculan todas las posibles vecindades para la solución *i* (en este caso *i* = 4). La solución del elemento cuatro se puede ver en la figura 4.12.

4.3.1. VECINDADES LÍTICAS POR INSERCIÓN

Estas vecindades son las que se generan al intentar insertar trabajos que no están en la solución. Las que se generan en este ejemplo para el elemento 4 de la población son las que se muestran en la figura 4.13.

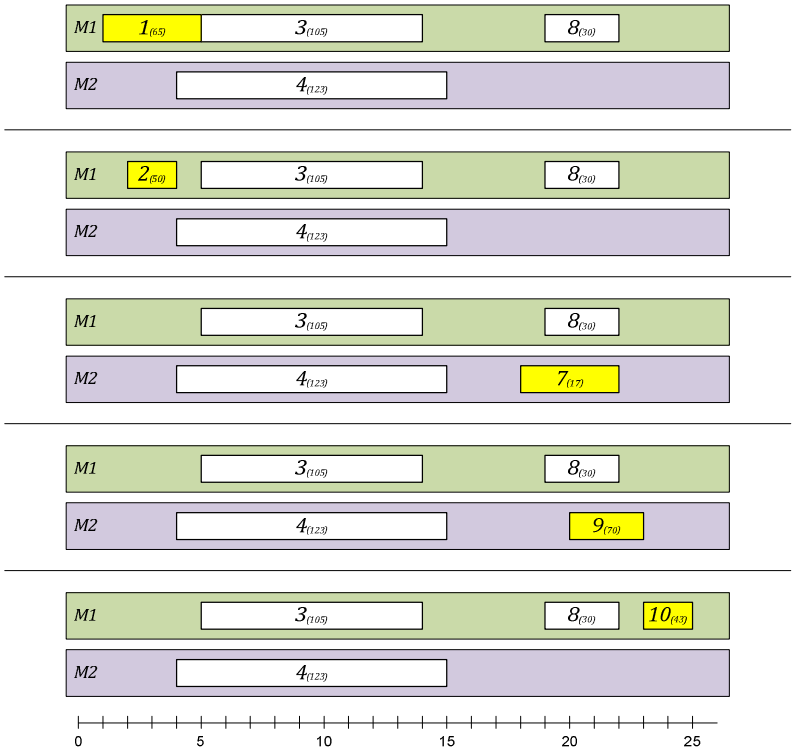

Figura 4.13. *Vecindades por inserción*

Los trabajos 5 y 6 no han podido insertarse, ya que en todos sus comienzos posibles coincidían con los trabajos 3 y 4. Por lo tanto 5 y 6 no han generado vecindad.

4.3.2. VECINDADES LÍTICAS POR SUSTITUCIÓN

En este procedimiento se sustituye cada trabajo de la solución por otro que no esté en ella. De todos los posibles trabajos entre los que elegir, se le da más probabilidad a los que tienen mayor peso. Las vecindades por sustitución que se crearían en este ejemplo son las que aparecen en la figura 4.14.

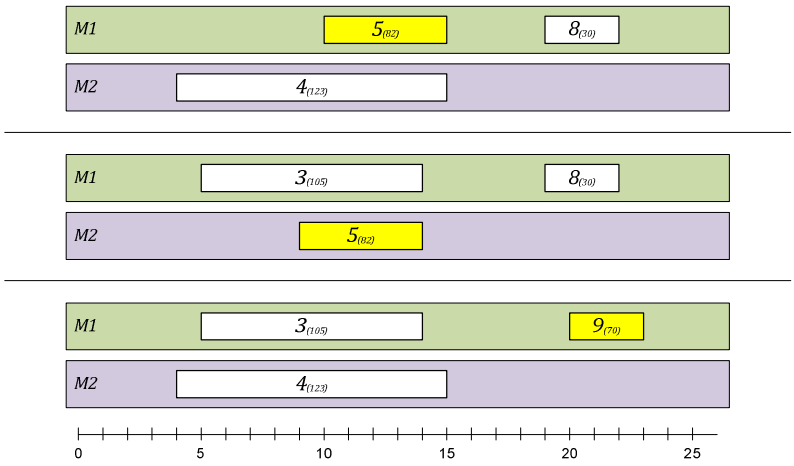

Figura 4.14. *Vecindades por sustitución*

Vemos que en las dos primeras vecindades, tanto al trabajo 3 como al 4 los ha sustituido el trabajo 5, que es, de los que podían sustituirlos, el que tiene un peso mayor.

4.3.3. VECINDADES LÍTICAS POR DESPLAZAMIENTO

Para terminar el cálculo de vecindades se desplazan los inicios de los trabajos existentes en la solución por su ventana de comienzos, para ver si pueden insertarse trabajos en el hueco que dejan. Al igual que antes, de los trabajos con posibilidad de insertarse en dicho hueco, se prioriza a aquellos con mayor peso.

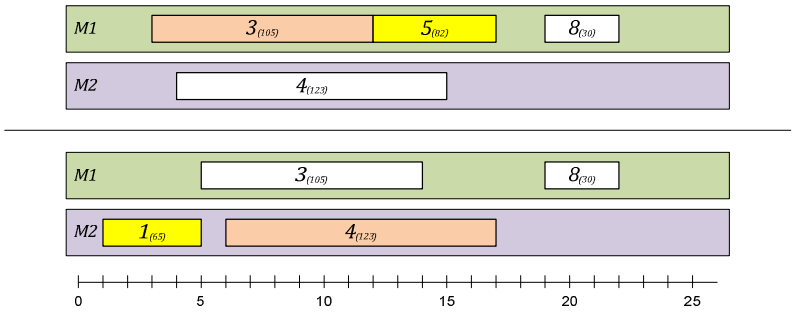

Figura 4.15. *Vecindades por desplazamiento*

En la figura 4.15 se puede ver cómo el trabajo 3 se ha desplazado para dejar hueco al 5 generando así una vecindad; y el trabajo 4 ha hecho lo mismo con el trabajo 1, obteniendo con ello una segunda vecindad. El trabajo 8 sin embargo al no tener movilidad (figura 4.10), no genera vecindad alguna.

Para concluir el proceso *lítico* se examinan todas las vecindades generadas (tabla 4.16) y se decide cuál será la que ocupe el lugar del elemento 4 según el valor de su solución.

	Trabajos (m,k)	NR	IT	Valor	LNR	LIT
Inserción	1(1,1) . 3(1,5) . 4(2,4) . 8(1,19)	0	0	323	0	0
	2(1,2) . 3(1,5) . 4(2,4) . 8(1,19)	0	0	308	0	0
	3(1,5) . 4(2,4) . 7(2,18) . 8(1,19)	0	0	275	0	0
	3(1,5) . 4(2,4) . 8(1,19) . 9(2,20)	0	0	328	0	0
	3(1,5) . 4(2,4) . 8(1,19) . 10(1,23)	0	0	301	0	0
Sustitución	4(2,4) . 5(1,10) . 8(1,19)	0	0	235	0	0
	3(1,5) . 5(2,9) . 8(1,19)	0	0	217	0	0
	3(1,5) . 4(2,4) . 9(1,20)	0	0	298	0	0
Desplazamiento	3(1,3) . 4(2,4) . 5(1,12). 8(1,19)	0	0	340	0	0
	1(2,1) . 3(1,5) . 4(2,6) . 8(1,19)	0	0	323	0	0

Figura 4.16. *Conjunto de vecindades*

Vemos que la mejor vecindad es la primera de las hechas mediante desplazamiento, pues el valor de su solución es el más alto (340).

Trabajos (m,k)	NR	IT	Valor	LNR	LIT
1(1,2) . 4(2,5) . 7(1,17)	0	IT_1	205	0	LIT_1
2(1,3) . 5(1,11) . 6(2,12) . 9(2,20)	NR_2	0	224	LNR_2	0
4(1,6) . 5(2,10) . 7(1,18) . 10(2,23)	0	LIT_3	265	0	LIT_3
3(1,5) . 4(2,4) . 8(1,19)	LNR_4	0	258	LNR_4	0
1(1,2) . 3(1,6) . 6(2,13) . 7(1,17)	NR_5	0	209	LNR_5	0

Trabajos (m,k)	NR	IT	Valor	LNR	LIT
1(1,2) . 4(2,5) . 7(1,17)	0	IT_1	205	0	LIT_1
2(1,3) . 5(1,11) . 6(2,12) . 9(2,20)	NR_2	0	224	LNR_2	0
4(1,6) . 5(2,10) . 7(1,18) . 10(2,23)	0	LIT_3	265	0	LIT_3
3(1,3) . 4(2,4) . 5(1,12). 8(1,19)	0	0	340	$LNR_{4\ (nuevo)}$	0
1(1,2) . 3(1,6) . 6(2,13) . 7(1,17)	NR_5	0	209	LNR_5	0

Figura 4.17. *Nueva población*

Escogida la vecina que va a sustituir al elemento *i*, se decide aleatoriamente su nuevo tipo, y se incluye en la población (figura 4.17). A continuación se

comprueba si la solución encontrada es la mejor hasta ese momento, y si lo es, se almacena como la mejor junto con la iteración en la que se ha obtenido. Con el proceso ya finalizado sólo queda asignarle un nuevo valor a su LNR_i o LIT_i según sea el caso (ecuaciones 3.1 y 3.2), y pasar a la siguiente iteración.

Suponemos que en la siguiente iteración se escoge el elemento 3 de la población, que es de tipo *lisogénico*, e igual que antes se comprueba si el número de iteraciones ha llegado al valor adecuado para realizar la mutación. Como $IT_3 = LIT_3$ sí se realiza la mutación. Una muestra de la solución que presenta este elemento se puede ver en la figura 4.18.

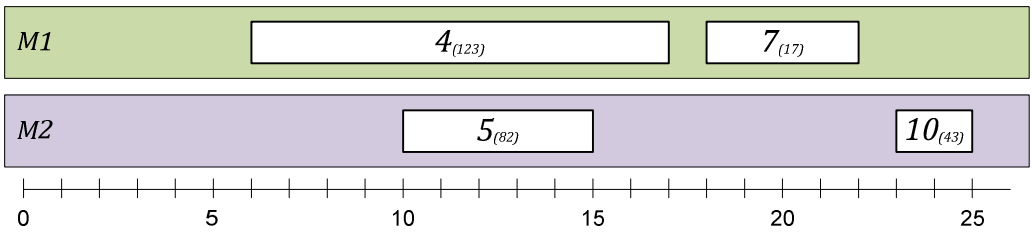

Figura 4.18. *Elemento 3 de la población*

En el proceso *lisogénico* no se calculan tantas vecindades como en el *lítico*. En este caso sólo se creará una, que será por inserción si el trabajo escogido de forma aleatoria no está en la solución, y por sustitución en caso contrario. Para esta ilustración vamos a ver ambos casos, pero en el algoritmo sólo se haría uno.

4.3.4. VECINDAD LISOGÉNICA POR INSERCIÓN

Si el trabajo escogido aleatoriamente no estuviese en la solución, se intenta insertar en ésta, generando así la vecindad si lo consiguiese. En caso de no poder incorporarse a la solución, no se generaría vecindad. Tomando por ejemplo el trabajo escogido fuese el 9, la vecindad quedaría como en la figura 4.19.

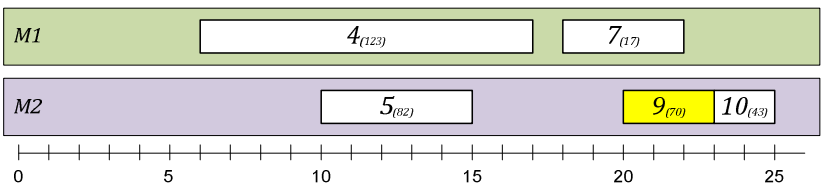

Figura 4.19. *Vecindad por inserción*

4.3.5. VECINDAD LISOGÉNICA POR SUSTITUCIÓN

En el caso en que el trabajo elegido sí que esté en la solución, entonces se intenta sustituir éste por otro que no esté, y al igual que en el caso *lítico*, para la sustitución del trabajo elegido tendrán preferencia aquellos trabajos cuyos pesos sean mayores.

Para el ejemplo suponemos que el trabajo elegido es el 4, y en ese caso la vecindad quedaría tal y como se muestra en la figura 4.20.

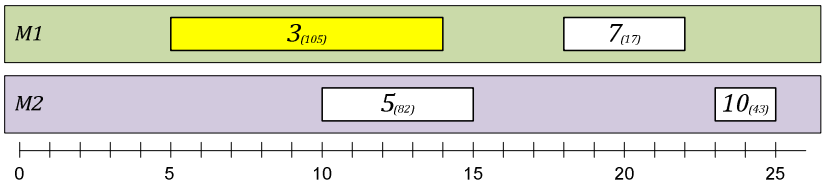

Figura 4.20. *Vecindad por sustitución*

En este caso siempre se genera vecindad, y si resulta que ningún trabajo puede sustituir al elegido, la vecindad sería igual al elemento original pero sin el trabajo elegido, permitiendo así que haya elementos que empeoren

Sea cual sea la vecindad generada, siempre se sustituye al elemento original por ella, aunque el valor de la solución de la vecindad sea inferior a la del elemento original. Y al igual que en el caso *lítico*, cuando se genera un nuevo elemento se comprueba si su solución es la mejor, y se le asignan nuevos tipo y valores de los parámetros correspondientes (LIT_3 o LNR_3).