Capítulo 6

Metaheurísticas Poblacionales

6.1. Introducción

Las metaheurísticas basadas en poblaciones o metaheurísticas poblacionales son aquéllas que emplean un conjunto de soluciones (población) en cada iteración del algoritmo, en lugar de utilizar una única solución como las metaheurísticas trayectoriales. Estas metaheurísticas proporcionan de forma intrínseca un mecanismo de exploración paralelo del espacio de soluciones, y su eficiencia depende en gran medida de cómo se manipule dicha población [32].

La figura 6.1 muestra gráficamente el funcionamiento de una metaheurística poblacional en problemas de optimización. A la izquierda, se representa la distribución de la población inicial sobre un espacio de búsqueda unidimensional mediante un conjunto de puntos que se distribuyen de forma prácticamente uniforme sobre el espacio. A la derecha, se representa la distribución de los individuos de la población en la última iteración del problema de maximización. En esta situación se puede apreciar que los individuos se agrupan alrededor del valor máximo. Este capítulo está dedicado a presentar las metaheurísticas poblacionales más conocidas en el ámbito de la optimización.

6.2. Algoritmos evolutivos

Los primeros trabajos científicos que se pueden encontrar en el campo de los *algoritmos evolutivos* (*Evolutionary Algorithms* - EA) provienen de finales de los años cincuenta. En esta década aparecen publicaciones [111][38][112][113] en las que se proponen EA sencillos para resolver problemas combinatorios. Ya en la década de los sesenta, L. Fogel [109], establece las bases de la programación genética, y en

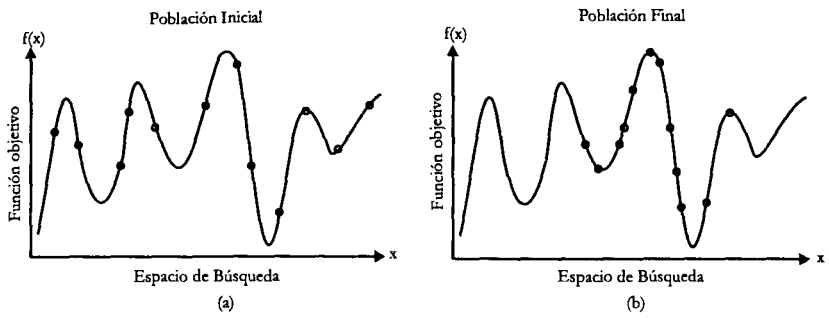

Figura 6.1: Poblaciones (a) inicial y (b) final durante un proceso de optimización utilizando una metaheurística poblacional. Generalmente, el espacio de búsqueda es multidimensional.

los setenta, Rechemberg [258] hace lo propio con las estrategias evolutivas. Holland [160] introduce los algoritmos genéticos a mediados de esta década. Estos trabajos (ordenados cronológicamente), suponen las bases de muchas metaheurísticas modernas como son los propios algoritmos genéticos [160][136], los algoritmos meméticos [222][224], los algoritmos culturales [268] o la inteligencia de enjambre [177] entre otros.

EA se basa en la idea *neo-darwiniana* de evolución de las especies, que se puede expresar en los siguientes términos:

Los individuos que tienen una mejor adaptación al medio tienen una probabilidad más elevada de vivir más tiempo, con lo que tendrán más posibilidades de generar descendencia que herede sus buenas características. En cambio, los individuos con peor adaptación al medio, tienen menos probabilidad de sobrevivir, por lo que tendrán menos oportunidades de generar descendencia y probablemente acaben extinguiéndose

Desde un punto de vista algorítmico, EA consta de los siguientes elementos [210][211]:

- **Población:** conjunto de soluciones candidatas de un problema dado, donde cada una de ellas recibe el nombre de individuo.
- **Selección:** mecanismo de elección de individuos sesgado de forma que sea más probable seleccionar a los mejores para que transmitan su contenido genético a su descendencia.

Algoritmo 6.1 Algoritmo Evolutivo

```
{x_best: TipoSolucion} = EA(N: integer; f: TipoFuncionObjetivo)
/*N es el número de individuos por generación*/
var
  x_g, x*_g: array [1,...,N] of TipoSolucion; // Población;

  begin
    g := 1; // Inicializar el número de generación
    {x} := inicializar(N); // Inicializar la población
    repeat
      {x*} := seleccionar(x, f); // Seleccionar los mejores individuos
      {x} := alterar(x*); // Alterar los individuos seleccionados
      g := g+1;
    until terminacion
    {x_best} := seleccionar(x, f); //Seleccionar la mejor solución encontrada
  end
```

- **Alteración:** mecanismo de generación de nuevos individuos mediante la modificación estocástica de los antiguos. En función del número de individuos antiguos utilizados, la transformación puede ser unaria (mutación) o de orden superior (cruce).

Estas características se pueden agrupar para diseñar un algoritmo de optimización robusto y eficaz. En el algoritmo 6.1 se presenta un pseudocódigo de alto nivel para esta metaheurística. EA inicializa una población de N individuos siguiendo algún determinado criterio. Esta población se modifica utilizando la aplicación de las operaciones de selección y alteración (típicamente, mutación y cruce) en sucesivas iteraciones hasta que se alcanza una condición de parada. Finalmente, el mejor individuo encontrado se devuelve como la solución al problema de optimización.

EA se organiza en cuatro grandes áreas que comparten la plantilla mostrada en el algoritmo 6.1, pero cuya implementación difiere sensiblemente [295]. Éstas son:

- **Programación Evolutiva** [108][109]**:** desarrollada en el marco de la predicción de secuencias de tareas. Se caracteriza por tener una representación muy ligada al dominio del problema. Habitualmente no tiene cruce, con lo que el mecanismo de mutación es más elaborado. El algoritmo inicializa una población para seleccionar todos los individuos como padres posteriormente. Después, muta los N padres para generar N hijos y evaluarlos. De la población de padres e hijos se seleccionan, de forma probabilística N hijos, permitiendo que el mejor individuo siempre sobreviva (esta estrategia se denomina "elitismo").

- **Estrategias de Evolución** [258][290]: desarrolladas en el ámbito de los problemas de optimización en hidrodinámica, por lo que utilizan como estructura de representación vectores reales. El algoritmo *inicializa* y *evalúa* una población para, posteriormente, *seleccionar* los padres de forma uniforme y aleatoria. La *combinación* de dos padres genera los hijos en una cantidad superior a los padres. Finalmente, se modifican los hijos utilizando una etapa de *mutación*. La supervivencia de los individuos es determinista y puede estar implementada con o sin elitismo.

- **Algoritmos Genéticos** [160][136]: se ha dedicado la sección 6.2.1 a su análisis debido a su gran relevancia en el contexto de la optimización.

- **Programación Genética** [182][183]: persigue la construcción de programas informáticos, utilizando un conjunto de programas generados aleatoriamente y los principios de EA.

EA constituye una herramienta de optimización muy versátil que se puede aplicar a una gran cantidad y variedad de problemas de optimización. EA siempre funciona relativamente bien, aunque para cada tarea puede existir una herramienta específica del problema que funcione mejor. En este sentido, su uso está especialmente indicado en los casos en que no se dispone de un conocimiento profundo del problema que se quiere resolver.

6.2.1. Algoritmos genéticos

Los *algoritmos genéticos* (*Genetic Algorithm* - GA) fueron presentados por J. Holland [160], aunque existen trabajos previos de otros científicos que desarrollaron ideas similares [111][38][112][113][109][258]. El concepto de recombinación de soluciones supone una de las aportaciones fundamentales de GA. Por otro lado, es también relevante la diferencia explícita entre la representación del problema (denominada genotipo), que habitualmente viene determinada por cadenas de bits conocidas como cromosomas, y las variables del problema en sí (denominadas fenotipo).

GA representa una metaheurística poblacional sencilla e intuitiva que, muy probablemente, sea la más utilizada. El principio de operación es el siguiente:

GA se basa en una población de soluciones candidatas, que evoluciona por medio de los mecanismos genéticos neo-darwinistas de selección, cruce y mutación

La nomenclatura utilizada en GA está muy relacionada con su inspiración biológica. En este sentido, es usual denominar "población de individuos" al conjunto de soluciones, de forma que cada individuo se corresponde con una solución. La solución en sí (variables del problema) establece el fenotipo del individuo, y su re-

presentación el genotipo. Esta representación recibe el nombre de cromosoma y cada uno suele estar compuesto por unidades discretas llamadas genes.

GA es un tipo de EA, y heredan sus características algorítmicas. Por esta razón, se ha omitido de esta sección el algoritmo que describe su pseudocódigo. Sin embargo, GA presenta una serie de particularidades que lo hace singular y con entidad propia como metaheurística. Los elementos que deben identificarse para resolver un problema con un algoritmo genético son:

- **Población inicial:** suele estar formada por una generación aleatoria de soluciones al problema dado [160][80][136].
- **Representación:** constituye una correspondencia entre las soluciones factibles (fenotipo) y la codificación de las variables (genotipo) [211]. Originalmente, las representaciones eran cadenas binarias [160][80][136]. Sin embargo, actualmente se han utilizado otras representaciones en problemas discretos [203], de permutaciones [261] y binarios [211].
- **Función de evaluación:** determina la calidad de los individuos de la población. Habitualmente, es una función monótona creciente que asigna un valor mayor cuanto mejor sea el individuo.
- **Operadores genéticos:** Son métodos probabilísticos que obtienen nuevos individuos. Suelen ser dependientes de la representación. Habitualmente, se utilizan los siguientes operadores:
  1. **Cruce:** consiste en la sustitución de un conjunto de genes de un padre por los genes correspondientes del otro padre para generar un nuevo individuo hijo. En [260] se puede encontrar diferentes métodos de cruce.
  2. **Mutación:** consiste en el cambio aleatorio de parte de un individuo. La mutación se emplea como mecanismo para preservar la diversidad de las soluciones y explorar nuevas zonas del espacio de soluciones.
- **Selección:** es un mecanismo que permite elegir con mayor probabilidad a los individuos que presenten un valor más elevado de la función de evaluación [211][260]. Este mecanismo suele estar implementado usando el método de la *ruleta*, del *torneo*, etc. En [260] se puede encontrar una descripción detallada de diferentes métodos de selección.

Es importante destacar que la cantidad de variantes que se pueden presentar en los algoritmos genéticos es enorme. En esta sección, sólo se han indicado las características básicas de esta metaheurística como son: la determinación de la población inicial; los métodos que se utilizan para inicializarla; las estrategias de selección, mutación y cruce; las probabilidades que se asignan a cada estrategia. En el caso de que

se quiera introducir cierto conocimiento del problema (representación de individuos a través de estructuras más complejas, operadores de cruce y mutación que tengan en cuenta estas estructuras, etc.), el diseño del algoritmo genético podría ser muy distinto. En [260] se apunta que prácticamente cada algoritmo genético es único.

Algoritmo 6.2 Algoritmo Memético

```
{x_best: TipoSolucion} = MA(N:integer; Operador: array [1...N] of TipoOperador; f: TipoFuncionObjetivo)
var
  x_g: array [1...N] of TipoSolucion; // Soluciones con peso
  g, i: integer;

  begin
    g := 1; // Inicializar el número de generación
    {x} := Inicializar(N); // Inicializar aleatoriamente la población
    for i := 1 to N do
      {x[i]} := OptimizadorLocal(x[i], Operador[i], f);
    end for
    repeat
      /*Paso generacional:*/
      {x} := Seleccionar(x, f); // Seleccionar los mejores individuos
      {x} := Alterar(x); // Reproducir nuevos individuos
      for i := 1 to N_s do
        {x[i]} := OptimizadorLocal(x[i], Operador[i], f);
      end for
      if convergencia(x) then
        {x} := Inicializar(N_s); // Inicializar aleatoriamente la población
        for i := 1 to N_s do
          {x[i]} := OptimizadorLocal(x[i], Operador[i], f);
        end for
      end if
      g := g + 1;
    until terminacion
    {x_best} := seleccionar(x, f); // Seleccionar la mejor solución encontrada
  end
```

6.2.2. Algoritmos meméticos

Los orígenes de los *algoritmos meméticos* (*Memetic Algorithms* - MA) se remontan a finales de los años ochenta, a pesar de que algunos trabajos en décadas anteriores también tienen similares características [225]. En aquella época, EA es-

taba comenzando a afianzarse sólidamente, una vez que el choque conceptual que había causado la introducción de estas técnicas en el mundo de la optimización se iba atenuando. Otro tanto se podía decir de técnicas relacionadas, como SA o TS. En general, estas técnicas hacían uso de heurísticas subordinadas para llevar a cabo el proceso de optimización. Fue en este escenario en el que surgió la idea básica que sustenta MA: combinar conceptos y estrategias de diferentes metaheurísticas para aunar las ventajas de las mismas.

MA debe su nombre al concepto de "*meme*" introducido por R. Dawkins [78] en una clara analogía con gen, para dar una explicación a la evolución cultural. El principio de operación de esta metaheurística es el siguiente:

De la misma forma que en una población se trasmiten los genes de los padres a los hijos, los memes se transmiten de cerebro a cerebro de la población

Desde un punto de vista algorítmico, es común considerar MA como un GA al que se añade un procedimiento de Búsqueda Local. Sin embargo, aunque MA toma ideas de ambos procedimientos, también se proponen nuevos operadores que los hacen diferentes e independientes de GA [71][225]. En este sentido, una de las características más descriptivas de MA es la incorporación de todo el conocimiento del problema que se tenga disponible, en contraposición a los algoritmos genéticos, que evitan, en la medida de lo posible, las particularidades del problema [71][224][225].

La algorítmica de esta metaheurística fue desarrollada por P. Moscato [222], en un trabajo donde se proponía la hibridación de un GA [160] con un procedimiento SA [180]. Actualmente se entiende que un MA está compuesto por una población de agentes, que son una extensión del individuo. Cada agente contiene una o varias soluciones y un procedimiento de mejora de estas soluciones [224]. Los agentes pueden mejorar durante su vida mediante procedimientos de búsqueda local. Además, se establecen relaciones de cooperación y competición con el resto de agentes de la población.

En el algoritmo 6.2 se presenta una plantilla general para los Algoritmos Meméticos. MA tiene una relación muy estrecha con EA, al menos desde un punto de vista sintáctico, debido fundamentalmente a su naturaleza poblacional [225]. Este hecho se ve reflejado en la similitud que existe entre este pseudocódigo y el presentado en el algoritmo 6.1. La diferencia fundamental entre ambos métodos se encuentra en la implementación de los grandes bloques de selección y alteración. La parte de competición viene determinada por los bloques de selección, con su correspondiente reemplazo de agentes. La cooperación entre agentes sustituye a las fases de alteración (mutación y cruce) en EA.

Una de las características más identificables de los algoritmos meméticos es el operador de optimización local cuyo esquema general se muestra en el algoritmo 6.3. Éste se define como un operador iterativo que actúa sobre un agente, de forma que

Algoritmo 6.3 Optimizador Local

```
{x: TipoSolucion} = OptimizadorLocal(x: TipoSolucion; Operador: TipoOperador; f: TipoFuncionObjetivo)
var
  x_nuevo: TipoSolucion;

  begin
    repeat
      {x_nuevo} := aplicar(x, Operador);
      if f(x_nuevo) > f(x) then
        x := x_nuevo;
      end if
    until Terminacion
  end
```

mejora la calidad de la solución o soluciones que contiene dicho agente hasta que no se pueda mejorar más.

Para una descripción exhaustiva de MA pueden consultarse los trabajos [224][225] [71] en los que se consideran descripciones y aplicaciones de esta metaheurística a distintos problemas de optimización.

6.3. Búsqueda dispersa

La *búsqueda dispersa* (*Scatter Search* - SS) es un procedimiento metaheurístico basado en formulaciones y estrategias introducidas durante la década de los setenta. A principios de esta década se propusieron los conceptos y principios fundamentales del método, basados en estrategias de combinación de reglas de decisión [198]. La primera descripción de la metaheurística aparece en [120], donde se establecen las líneas básicas de SS, pero hasta 1998 [124] no se constituye la forma y terminología definitiva del método.

Los antecesores de SS fueron aplicados inicialmente a problemas de planificación de tareas (*scheduling*) [119], donde se proponían una serie de estrategias para combinar las reglas de decisión y restricciones [120]. Estos métodos obtenían nuevas reglas mediante la combinación lineal de otras [119]. Los estudios experimentales demostraron que esta combinación producía mejores resultados que la aplicación de las reglas originales por separado.

El principio de operación de esta metaheurística puede ser expresado del siguiente modo [196]:

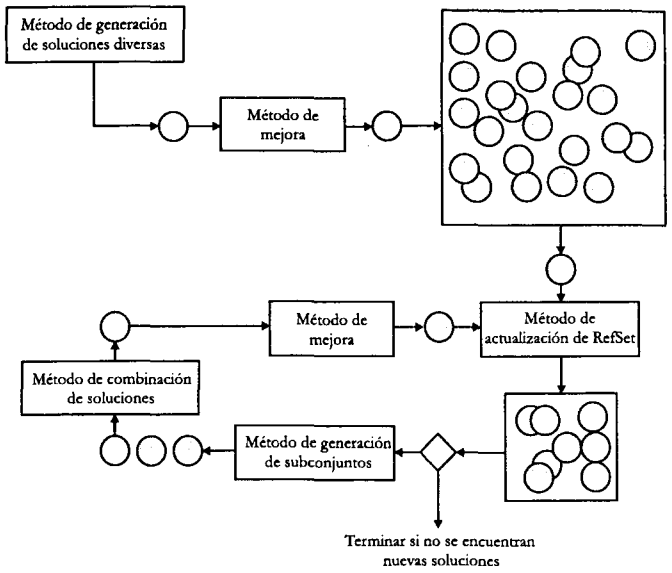

Figura 6.2: Representación gráfica de las etapas de la búsqueda dispersa (adaptado de [198]).

La información sobre la calidad o el atractivo de un conjunto de reglas, restricciones o soluciones se puede utilizar mediante la combinación de éstas en lugar de aisladamente. De este modo, dadas dos soluciones de un problema, se podría obtener, mediante su combinación, una nueva solución que mejore a las que la originaron

El motor fundamental de SS consiste en la *combinación de soluciones* de alta calidad, con el objetivo de obtener mejores soluciones que las originales. En este sentido, se dice que SS es un Algoritmo Evolutivo, ya que mantiene una población de soluciones en cada momento [198]. Sin embargo, a diferencia de éstos, SS no está fundamentado en la aleatorización sobre un conjunto grande de soluciones, sino en elecciones sistemáticas y estratégicas sobre un conjunto pequeño. En otras palabras, SS está diseñada para trabajar con un conjunto de soluciones llamado conjunto de referencia (*RefSet*), caracterizado por contener "buenas" soluciones. En el contexto de SS, se consideran buenas soluciones las que o bien tienen alta calidad (dada por la función objetivo) o bien tienen alta diversidad[1] (muy alejadas de las soluciones de alta calidad seleccionadas).

[1]Es importante notar que el cálculo del subconjunto de las soluciones más diversas de un conjunto es un problema difícil.

Las combinaciones de soluciones son versiones generalizadas de combinaciones lineales (en el espacio euclídeo). *A priori*, para muchos problemas combinatorios no tiene sentido la combinación lineal de soluciones. Por ejemplo, en el caso de problemas cuya solución es una permutación (como el Problema del Viajante de Comercio o el Problema del Enrutamiento de Vehículos) no se pueden combinar linealmente dos soluciones. Para aplicar SS a este tipo de problemas, se extiende el concepto de combinación lineal, introduciendo los cruces o combinaciones basados en votos (*voting method*) [186]. En ellos, cada solución (formada por componentes) "vota" para que sus componentes aparezcan en la nueva solución generada. En [198] se resalta que esta estrategia es una de las claves del éxito de SS. Además, se fuerza a que el resultado de dicha combinación satisfaga ciertas condiciones de factibilidad [120].

Una característica notable de SS es la integración de métodos de mejora y de combinación de soluciones. Los procedimientos de mejora suelen ser algoritmos sencillos de Búsqueda Local, pero no existe ninguna limitación para utilizar otros métodos más complejos como la Búsqueda Tabú [121]. La aplicación de esta etapa asegura que la calidad de las soluciones consideradas siempre es elevada, al menos en una vecindad dada.

Desde un punto de vista algorítmico, la metaheurística SS consiste básicamente en cinco procedimientos, que se representan gráficamente en la figura 6.2:

1. **Método de generación de soluciones diversas (*Diversification Generation Me- thod*):** genera una población de N soluciones diversas (típicamente $N = 100$). Cada una de estas soluciones se optimiza utilizando el método de mejora. A continuación, se extrae un subconjunto pequeño de buenas soluciones (usualmente alrededor de 10). Este subconjunto recibe el nombre de conjunto de referencia (*Reference Set - RefSet*).

2. **Método de mejora (*Improvement Method*):** transforma las soluciones de prueba en una o más soluciones de prueba mejoradas. Típicamente se trata de un procedimiento de Búsqueda Local, aunque se puede utilizar cualquier metaheurística trayectorial. El método debe permitir, a partir de una solución no factible, obtener una que lo sea y después intentar mejorar su calidad. Si el método no logra mejorar la solución inicial se considera que el resultado es la propia solución inicial.

3. **Método de actualización del conjunto de referencia (*RefSet Update Method*):** construye y mantiene el conjunto de referencia (*RefSet*), compuesto por las b ($b << N$) mejores soluciones encontradas hasta el momento. A partir del conjunto de soluciones iniciales se extrae el conjunto de referencia siguiendo un criterio de convivencia entre soluciones de calidad y soluciones diversas. Éstas son ordenadas de mayor a menor calidad.

Algoritmo 6.4 Búsqueda Dispersa

```
{RefSet[1]: TipoSolucion} = SS(N, b, SubsetSize: integer; f: TipoFuncionObjetivo)
var
  DiverseSet: array [1...N] of TipoSolucion;
  RefSet: array [1...b] of TipoSolucion;
  p_nueva: TipoSolucion;
  P: array [1...N,1...SubsetSize] of integer;
  NewSolution: boolean;
  i: integer;

  begin
    {DiverseSet} := GenerarDiversas(N); // Incluyendo mejora de las soluciones
    /*Inicializa RefSet con los b/2 mejores individuos y los b/2 más diversos en
    DiverseSet*/
    {RefSet} := ConstruirRefSet(DiverseSet, b, f);
    NewSolution := TRUE;
    while NewSolution do
      NewSolution := FALSE;
      {P} := GenerarGrupos(SubsetSize);
      while P ≠ ∅ do
        {p_nueva} := Combinar(RefSet, P[1,:]);
        P := P[2:,:]; // Eliminar de P la combinación utilizada
        {p_nueva} := Mejorar(p_nueva, f); // Mejorar las soluciones nuevas
        if f(p_nueva) > f(RefSet[b]) then
          {RefSet} := Actualizar(RefSet, p_nueva); // Actualizar RefSet
          NewSolution := TRUE;
        end if
      end while
    end while
  end
```

- **Creación:** Típicamente, el *RefSet* se inicializa con las $b/2$ mejores soluciones del conjunto diverso. El resto ($b/2$) se extraen con el criterio de maximizar la distancia con las ya incluidas en el *RefSet*. Para esta selección, se debe definir una función distancia que dependerá del problema.
- **Actualización:** Las nuevas soluciones generadas, fruto de las combinaciones, que mejoren a las del *RefSet* deberán reemplazarlas. El conjunto de referencia mantiene así un tamaño constante b, pero la calidad de las soluciones que contiene mejora durante la búsqueda. Fundamentalmente, existen dos estrategias para tratar las soluciones obtenidas por medio de la combinación. La primera de ellas, denominada "estática", almacena todas las soluciones temporalmente hasta que se han combinado todas. Después, se analizan y se decide si alguna de las soluciones generadas debe entrar en el conjunto de referencia. En la estrategia "dinámica", la actualización del *RefSet* se realiza en el momento en que se produzca una solución mejor que alguna de las ya almacenadas.

4. **Método de generación de subconjuntos (*Subset Generation Method*):** SS se basa en la combinación exhaustiva de todas las soluciones del *RefSet*. Este método especifica la forma en que se construyen los subconjuntos a los que posteriormente se aplica el método de combinación. Una implementación sencilla y utilizada frecuentemente consiste en restringir los conjuntos a parejas de soluciones. El método de combinación de soluciones se aplican a todas y cada una de ellas.
5. **Método de combinación de soluciones (*Solution Combination Method*):** combina las soluciones de los subconjuntos generados en una o más soluciones.

En el algoritmo 6.4 se muestra un pseudocódigo de alto nivel de SS muy similar al propuesto por M. Laguna y R. Martí en [186]. SS establece una metodología muy flexible y modular, por lo que es relativamente sencillo introducir extensiones de los procedimientos que forman parte de la metaheurística. En [186][198][128][129][131][132] se describen diversas modificaciones en los procedimientos de SS que han aumentado el rendimiento de la metaheurística. Algunas de las mejoras más extendidas son las siguientes:

- **Reconstrucción del conjunto de referencia:** se utiliza cuando no se pueden generar nuevas soluciones que mejoren las que ya existen en el *RefSet*. En esta situación, el conjunto de referencia es parcialmente reconstruido introduciendo soluciones que aporten diversidad, sustituyendo, típicamente, $b/2$ soluciones por otras tantas que aportan diversidad.
- **Memoria:** la memoria basada en frecuencia suele mejorar el comportamiento de la metaheurística, debido fundamentalmente a que se asegura la diversifica-

ción. Para ello, se generan soluciones que contienen componentes significativas. Con esta estrategia se puede evitar que aparezcan soluciones con componentes repetidas o que hayan conducido a soluciones de baja calidad.

- **Tamaño del conjunto de referencia:** no existe un tamaño óptimo para el *RefSet*, por lo que puede ser una buena estrategia aumentar el tamaño cuando no se consiguen soluciones aceptables.
- **Fase de mejora selectiva:** en muchas ocasiones, aplicar la fase de Búsqueda Local a todas las combinaciones no asegura que se encuentren soluciones de mayor calidad. Por esta razón, puede ser útil restringir la aplicación de la Búsqueda Local a soluciones que mejoren un cierto umbral de calidad (soluciones "prometedoras").
- **Equilibrio entre intensificación y diversificación:** se debe hacer un balance entre el tiempo que se emplea en buscar soluciones de alta calidad (intensificación) y en explorar el espacio de soluciones, (diversificación). Por ejemplo, un aumento de b, incrementa la diversificación del procedimiento, pero habitualmente también crece el coste computacional.
- **Métodos de combinación distintos:** el uso de métodos de combinación de soluciones distintos permite explorar regiones diferentes del espacio de búsqueda. Algunos trabajos proponen la combinación de elementos aleatorios con deterministas, de tal forma que la metaheurística decide utilizar uno u otro en función del éxito que haya tenido el proceso de búsqueda con cada uno de ellos.

El método SS ha sido aplicado con éxito a una cantidad importante de problemas de optimización. En [186] aparecen aplicaciones de esta metaheurística, además de un conjunto de referencias que facilitan su implementación para problemas concretos.

6.4. Reencadenamiento de trayectorias

El método *reencadenamiento de trayectorias* (*Path Relinking* - PR) fue originalmente propuesto como una estrategia que integra procesos de intensificación y diversificación en el contexto de TS [127]. Es un método que está en desarrollo y su propuesta es muy reciente. PR tiene una gran relación con SS [128], descrito en la sección anterior. De hecho, el establecimiento de la forma y terminología de SS y PR se establece conjuntamente en un trabajo de F. Glover en 1998 [124]. El principio de operación que sigue PR es el siguiente:

PR genera nuevas soluciones mediante la exploración de trayectorias en el espacio de soluciones. Para ello, partiendo de dos soluciones de alta calidad, una

Algoritmo 6.5 Reencadenamiento de Trayectorias

```
{RefSet[1]: TipoSolucion}=PR(N, b, NumImp, SubsetSize: integer; f: TipoFuncionObjetivo)
var
  DiverseSet: array [1...b] of TipoSolucion;
  RefSet: array [1...N] of TipoSolucion;
  Ruta: list of TipoSolucion;
  p_inicial, p_guia): TipoSolucion;
  P: array [1...N,1...SubsetSize] of integer;
  NewSolution: boolean;
  i: integer;

  begin
    {DiverseSet} := GenerarDiversas(N); // Incluyendo mejora de las soluciones
    {RefSet} := ConstruirRefSet(DiverseSet,b,f); // con los mejores individuos
    NewSolution := TRUE;
    while NewSolution do
      NewSolution := FALSE;
      {P} := GenerarGrupos(SubsetSize);
      while P ≠ ∅ do
        {p_inicial, p_guia} := Seleccionar(RefSet,P[1,:]); // Selecc. una pareja de P
        P := P[2:,:]; // Eliminar de P la combinación utilizada
        {Ruta} := Reencadenamiento(p_inicial, p_guia); // Generar nuevas soluciones
        for i := 1 to Nelem(Ruta)/NumImp do
          {Ruta[i]} := Mejorar(Ruta[i]); // Mejorar soluciones nuevas
        end for
        for i := 1 to Nelem(Ruta) do
          if f(Ruta[i]) > f(RefSet[b]) then
            {RefSet} := Actualizar(RefSet,Ruta); // Actualizar RefSet
            NewSolution := TRUE;
          end if
        end for
      end while
    end while
  end
```

actuando como origen y la otra como destino, se genera un camino en el espacio de soluciones entre origen y el destino

Se suele considerar PR como una extensión del mecanismo de combinación de soluciones de SS [127]. El método de combinación en PR se basa en la generación de trayectorias entre soluciones en el espacio de búsqueda, en lugar de llevar a cabo combinaciones lineales entre ellas. En otras palabras, dadas dos buenas soluciones x_1 y x_2, PR construye un camino entre ambas, comenzando por la solución x_1, denominada "solución de partida" y, llevando a cabo una serie de movimientos, intenta llegar a la solución x_2, denominada "solución guía". PR construye la trayectoria $x_1 \rightarrow x_2$ eliminando paulatinamente los atributos de la solución de partida para introducir atributos que pertenecen a la solución guía. El objetivo es encontrar en estos caminos soluciones que mejoren a aquéllas que originaron la trayectoria.

Desde un punto de vista algorítmico, PR comparte con SS los 5 procedimientos generales expuestos en la sección 6.3 (ver figura 6.2). De este modo, PR y SS son métodos que operan sobre un conjunto de soluciones de referencia y difieren en la forma en la que el conjunto de referencia es construido, mantenido, actualizado y mejorado [143]:

1. **Método de generación de soluciones diversas (*Diversification Generation Method*):** Su implementación es equivalente a la expuesta en la sección 6.3.
2. **Método de mejora (*Improvement Method*):** transforma las soluciones de prueba en una o más soluciones de prueba mejoradas. Al contrario que sucede en el esquema de SS, en PR se suele aplicar el procedimiento de mejora únicamente a un subconjunto de tamaño *NumImp* de las soluciones generadas. Usualmente se utiliza un procedimiento de Búsqueda Local, aunque se puede utilizar una metaheurística.
3. **Método de actualización del conjunto de referencia (*RefSet Update Method*):** construye y mantiene el conjunto de referencia (*RefSet*), compuesto por las b mejores soluciones encontradas hasta el momento. A partir del conjunto de soluciones iniciales se extrae el *RefSet* siguiendo un criterio de convivencia entre soluciones de calidad y diversidad. Éstas son ordenadas de mayor a menor calidad.
   - **Creación:** El *RefSet* se inicializa con las b mejores soluciones en el conjunto diverso.
   - **Actualización:** Las soluciones fruto de las combinaciones que mejoren aquéllas en el *RefSet* deberán reemplazarlas. El conjunto de referencia mantiene así un tamaño constante b, pero la calidad de las soluciones que contiene mejora durante la búsqueda.

4. **Método de generación de subconjuntos (*Subset Generation Method*):** PR se basa en la generación de trayectorias entre conjuntos (típicamente parejas) de soluciones de calidad. La generación de estas trayectorias es aplicado a todos y cada uno de los subconjuntos generados.
5. **Método de combinación de soluciones (*Solution Combination Method*):** combina las soluciones de los subconjuntos generados en una o más soluciones. Para parejas, el método de combinación de PR elige dos soluciones del *Ref-Set* $x_{inicial}$ y x_{guia}, y construye un camino entre ambas, comenzando por $x_{inicial}$ y llevando a cabo una serie de movimientos (que generan nuevas soluciones) para llegar a la solución x_{guia}. Este procedimiento se implementa de forma distinta si las trayectorias se generan a partir de grupos de soluciones de mayor tamaño.

En el algoritmo 6.5 se muestra un pseudocódigo de esta metaheurística para el caso más usual en el que se emplean dos soluciones para generar cada ruta. PR es una metaheurística que, al igual que SS, permite introducir modificaciones en sus procedimientos. En [198][128][129][131][132] se presentan diversas modificaciones que han mejorado sustancialmente el rendimiento de la metaheurística para algunos problemas:

- **Variación:** consiste en generar trayectorias desde la solución de arranque hasta un punto intermedio y simultáneamente desde la solución guía hasta otro punto intermedio. Finalmente se construye la trayectoria entre los dos puntos intermedios.
- **Tunelación:** permite la variación en la estructura de la vecindad. El uso de esta estrategia permite, por ejemplo, que el proceso de búsqueda considere soluciones pertenecientes a una región no factible. Es importante notar que la búsqueda no puede quedar atrapada en una solución no factible, ya que la solución guía siempre lo es.
- **Reencadenamiento extrapolado:** genera trayectorias entre dos soluciones más allá de los puntos intermedios entre ellas, extendiendo el camino que las une.
- **Padres múltiples:** utiliza un conjunto de soluciones guía en lugar de una sola. En este contexto, el siguiente movimiento desde la solución de partida x_1 viene determinado por la influencia de todas componentes presentes en los padres, de forma que aquellos padres con una componente más adecuada respecto a la función objetivo, guiarán la formación de la trayectoria.
- **Vecindades constructivas:** comienza con una solución de partida incompleta o incluso nula para generar una trayectoria influida por un conjunto de soluciones. La solución se construye mediante movimientos hacia las soluciones guía,

introduciendo progresivamente sus elementos. Generalmente, la inclusión de características en la solución se realiza utilizando el mecanismo de votos.

6.5. Algoritmos de estimación de la distribución

Los *algoritmos de estimación de la distribución* (*Estimation of Distribution Algorithms* - EDA) [187][194] son EA que usan una colección de soluciones candidatas para explorar trayectorias de búsqueda evitando los óptimos locales. Es una metaheurística actualmente en desarrollo, propuesta por Mühlenbein y Paass en 1996 [208], aunque existen trabajos previos de Zhigljavsky [334]. Anteriormente, Holland [160] ya había considerado la posibilidad de considerar la interacción entre variables para incorporarla en el contexto de los GA. La aparición de EDA estuvo fundamentalmente motivada por los siguientes hechos [188]:

- El rendimiento de EA depende de la elección de los valores de varios parámetros asociados. La determinación de estos valores constituye un problema de optimización en sí mismo.
- La predicción de los movimientos de la población sobre el espacio de búsqueda es muy compleja.
- El procedimiento de cruce en los algoritmos genéticos puede destruir ciertas agrupaciones de componentes de alta calidad.

El principio de funcionamiento de EDA viene dado por:

El uso de la estimación y posterior muestreo de una distribución de probabilidad aprendida a partir de los individuos seleccionados permite sustituir el cruce y la mutación en el contexto de los Algoritmos Evolutivos

EDA es una metaheurística que generaliza GA, sustituyendo el cruce y la mutación por el aprendizaje y muestreo de una distribución de probabilidad, que describe a los mejores individuos de la población en cada iteración. En este sentido, la característica principal de EDA es que no utiliza directamente los individuos, sino una determinada función de distribución de probabilidad. Utilizando la estimación de la distribución, EDA captura y explota eficientemente las relaciones existentes entre las variables involucradas en el dominio del problema. Por medio de esta estrategia, EDA intenta evitar la destrucción de individuos de alta calidad, al mismo tiempo que sortea la determinación de parámetros necesaria en la computación evolutiva [142].

Desde un punto de vista algorítmico, un EDA considera cuatro etapas, que se disponen como muestra el algoritmo 6.6:

Algoritmo 6.6 Algoritmos de Estimación de la Distribución

```
{x_best: TipoInidividuo} = EDA(N: integer; f: TipoFuncionObjetivo)
var
  poblacion: array [1...N] of TipoInidividuo; // Soluciones con peso
  g: integer; // generación
  M: integer;
  p_g(x): ⟨Función de distribución⟩;

  begin
    g := 1; // Inicializar el número de generación
    {poblacion} := Inicializar(N); // Inicializar aleatoriamente la población
    repeat
      /*Seleccionar M individuos de acuerdo a un mecanismo de selección*/
      {poblacion} := Seleccionar(poblacion);
      /*Estimar la distribución de probabilidad de los individuos seleccionados*/
      {p_g(x)} := EstimarPDF(p(x|poblacion), f);
      /*Muestrear N individuos*/
      ⟨poblacion ~ p_g(x)⟩;
      g := g + 1;
    until terminacion
    {x_best} := SeleccionarMejor(poblacion, f);
  end
```

- **Inicializar:** la generación de la población inicial se realiza aleatoriamente o utilizando cualquier procedimiento heurístico.
- **Seleccionar:** genera un subconjunto de individuos de alta calidad de la población anterior. El método de selección suele elegir con más probabilidad a los individuos con pesos mayores. Un ejemplo típico puede ser el método de la ruleta [210][211].
- **Estimar:** estima la función de distribución de probabilidad $p_g(x)$ a partir de los individuos seleccionados.
- **Muestrear:** genera una nueva población mediante el muestreo de la función de distribución $p_g(x)$ obtenida en el paso anterior.

La estimación de la función de distribución es la fase más complicada de este enfoque. Se realiza a partir de un subconjunto de "buenos individuos" de la población inicial. La función de distribución estimada expresa la interrelación existente entre las componentes. En este sentido, la función de distribución es tanto más difícil de modelar cuanto más complicada sea la dependencia entre las componentes. En el

caso más sencillo, el individuo x^i está compuesto por n^i atributos independientes. En el caso más general, $p_g(x)$ es una función de distribución conjunta de dimensión n que no se puede estimar de forma exacta. Para un valor de n suficientemente grande el problema es inabordable. La forma de tratar este problema consiste en suponer que la función de distribución de probabilidad se puede factorizar de acuerdo a un modelo simplificado. A continuación, se describen brevemente algunas de las propuestas de factorización de dicha función de distribución conjunta:

- **Sin dependencias:** se supone que la función de distribución de probabilidad se puede factorizar como el producto de funciones de probabilidad univariantes e independientes.
- **Dependencias bi-variadas:** se considera que existe una dependencia entre pares de componentes, de tal forma que, sólo se tienen que considerar estadísticos de orden 2. En [37][13][243] se pueden encontrar trabajos en esta línea.
- **Dependencias múltiples:** este caso es el más general de todos. Se requieren estadísticos de orden superior a dos. Generalmente, este problema se aborda mediante redes bayesianas. Algunas implementaciones de este estilo se pueden encontrar en [155][187].

6.6. Algoritmos culturales

Los *algoritmos culturales* (*Cultural Algorithms* - CA) son un tipo de EA que se inspiran en los teorías y modelos que proponen algunos sociólogos y arqueólogos para intentar explicar la evolución cultural. CA fue introducido por Reynolds [265][267][268] a finales de la década de los setenta como vehículo para modelar la evolución y el aprendizaje social. Consisten en una población evolutiva cuyas experiencias se integran en un espacio de creencias que consiste en varias formas de conocimiento simbólico. Se han aplicado con éxito en un conjunto diverso de áreas [281]. En [268] se expone una revisión de los modelos utilizados a lo largo de la historia para explicar la evolución cultural.

El principio de funcionamiento de esta metaheurística es el siguiente:

Los algoritmos culturales establecen que la toma de decisiones depende de la cultura heredada, así como de las creencias de la población actual

En las sociedades humanas, la cultura puede ser considerada como un contenedor de información pasada, en el sentido de una "memoria histórica". La cultura es independiente de los individuos que la generaron y potencialmente accesible por todos los miembros de la sociedad. Este esquema se puede utilizar para desarrollar estrategias

que permitan encontrar soluciones a un problema dado, modelando las relaciones entre los individuos, la sociedad y la cultura de dicha sociedad. Los sistemas culturales se pueden interpretar como un mecanismo de herencia en dos niveles:

1. **Nivel micro-evolutivo:** el contenido genético pasa de los padres a los hijos.
2. **Nivel macro-evolutivo:** el contenido cultural pasa de una generación a otra.

En cada iteración, la mejora genética de la población y las creencias adquiridas, se transmiten a la siguiente generación [92]. CA utiliza esta información para guiar a los individuos, de modo que pueda distinguirse entre soluciones de alta y de baja calidad.

En el contexto de los algoritmos culturales, se produce una evolución tanto en la sociedad, compuesta por individuos, como en la cultura, compuesta por creencias. En este sentido, se puede considerar que los algoritmos culturales están estructurados en las siguientes componentes:

- **Población:** que evoluciona mediante los mecanismos de selección, reproducción y evaluación.
- **Espacio de creencias:** permite guiar la búsqueda mediante el conocimiento adquirido durante la resolución del problema.
- **Protocolo de comunicación:** describe la interacción entre la población y sus creencias. Considera dos tipos de relaciones:
  - establecimiento del conjunto de individuos (aceptables) que pueden influir en la modificación de las creencias adquiridas.
  - establecimiento del mecanismo de influencia de las creencias sobre los individuos.

Reynolds [268], establece que un algoritmo cultural viene descrito por un conjunto de ocho componentes:

$$CA = [P, S, V_c, f, B, Aceptar, Ajustar, Influir] \tag{6.1}$$

donde

- P es la población de individuos que compone la sociedad,
- S es el operador de selección,
- V_c es un operador de variación,
- f es la función objetivo,

Algoritmo 6.7 Algoritmo Cultural

```
{x_best: TipoSolucion} = CA(N: integer; f: TipoFuncionObjetivo)
var
  P: array [1...N] of TipoSolucion; // Soluciones con peso
  B: ⟨ Creencias ⟩;
  g: integer; // generación

  begin
    g := 1; // Inicializar el número de generación
    {P} := InicializarPoblacion(N); // Inicializar los elementos de la población
    {B} := InicializarCreencias; // Inicializar las creencias de la población
    repeat
      {P} := Evaluar(P,f); // Evaluar la calidad de los individuos
      {B} := Ajustar(B,Aceptar(P)); // Ajustar las creencias con los indiv. influyentes
      {P} := Modificar(P,influir(B)); // Modificar los individuos con las creencias
      g := g + 1;
      {P} := Seleccionar(P,f);
    until terminacion
    {x_best} := SeleccionarMejor(P,f); // Seleccionar la mejor solución encontrada
  end
```

- B es el espacio de creencias,
- *Aceptar* es la función de aceptación y determina el protocolo de comunicación entre el espacio de creencias y la sociedad (en el sentido *sociedad* $\rightarrow$ *creencias*),
- *Ajustar* es un operador del espacio de creencias que actualiza las creencias e
- *Influir* es un conjunto de funciones de influencia que afectan al operador de variación V_c. Determinan el protocolo de comunicación entre el espacio de creencias y la sociedad (en el sentido *sociedad* $\leftarrow$ *creencias*).

En las versiones más sencillas de CA [265], la primera componente se corresponde con una implementación de un algoritmo genético cualquiera, de tal forma que el mecanismo de selección S y de variación (cruce y mutación) V_c equivalen a sus correspondientes genéticos. En estos primeros modelos [266], el espacio de creencias contiene únicamente un operador cultural que aprende la idoneidad de usar un determinado operador genético sobre la población, de forma que la probabilidad de aplicarlo aumenta o disminuye en función del efecto producido sobre la calidad

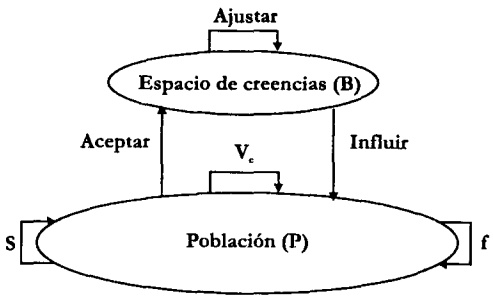

Figura 6.3: Esquema general para un algoritmo cultural.

de los individuos. En este modelo, la función de aceptación es del 100 %, de modo que se acepta cualquier mejora. Además se permite que cualquier individuo pueda modificar el espacio de creencias.

En una segunda evolución de la metaheurística, se introduce en el espacio de creencias un conocimiento sobre las estructuras de alta calidad que aparecen en la población [303]. En este modelo, la función de aceptación es del orden del 50 %, con lo que no todos los individuos pueden modificar el espacio de creencias. La función de influencia modifica la calidad del individuo en lugar de su estructura.

En la tercera versión se generaliza el algoritmo para poder abordar problemas de optimización de funciones reales. En esta versión, la implementación del espacio de soluciones es bastante más compleja. En [268] se puede encontrar una descripción detallada de este método.

En la cuarta y última generación de CA, tanto la población como el espacio de creencias utilizan estructuras de subgrafos. Éstas, se interpretan como estructuras de decisión que interactúan entre sí para encontrar una solución al problema. El espacio de creencias está formado por estructuras de subgrafo, que se caracterizan por aparecer en los individuos válidos. En este caso, la función de aceptación es del orden del 20 %. La función de influencia se diseña de tal forma que se reduzca la probabilidad de que el cruce pudiera destruir sub-estructuras que aparecen en el espacio de creencias.

6.7. Inteligencia de enjambre y opt. por enjambre de partículas

La *inteligencia de enjambre* (*Swarm Intelligence* - SI) se inspira en la forma colectiva de actuar de sociedades muy poco complejas, compuestas por individuos muy poco sofisticados. En la naturaleza se pueden encontrar numerosos ejemplos de este tipo de sociedades, como los bancos de peces, las colonias de hormigas o las bandadas de pájaros, que se comportan como si fueran un único individuo. Actualmente, se estudian los modelos biológicos de estos enjambres para entender cómo actúan, consiguen objetivos o evolucionan. Este enfoque general se ha concretado en la *optimización por colonias de hormigas* (*Ant Colony Optimization* - ACO) propuesta por M. Dorigo en 1992 [82] (ver sección 7.5) y la *optimización por enjambre de partículas* (*Particle Swarm Optimization* - PSO), propuesta por J. Kennedy y R. Eberhart en 1995 [175].

El principio de operación en el que se basa SI es el siguiente:

La inteligencia de enjambre es una propiedad que poseen ciertos sistemas biológicos compuestos por agentes sencillos (hormigas, peces, pájaros, etc.) en los que el comportamiento colectivo viene descrito por la interacción local entre agentes, de tal forma que se obtiene un funcionamiento global coherente

Desde un punto de vista algorítmico, un enjambre está compuesto por agentes que establecen relaciones de cooperación para conseguir un objetivo determinado. Cada agente utiliza un conjunto de reglas sencillas (locales) de forma relativamente independiente a los demás (salvo la cooperación entre agentes próximos), donde no existe un líder que determine la estrategia a seguir. En estas condiciones, emerge una inteligencia colectiva del conjunto de agentes, dando lugar a fenómenos de autoorganización. Aunque los agentes sean simples, el resultado de su interacción global puede llegar a ser muy complejo (como la realización de movimientos prácticamente al unísono o estrategias de protección frente a depredadores). En el algoritmo 6.8 se muestra un pseudocódigo de alto nivel para PSO.

PSO se inspira en la sociabilidad de los individuos que componen un sistema biológico [176]. La exploración del espacio de búsqueda se realiza utilizando una población de individuos llamados partículas. Las partículas se distribuyen sobre el espacio de búsqueda, de forma que su posición, determinada por unas coordenadas $\overrightarrow{x}^i$, representa los valores que toman las variables de decisión del problema. Cada partícula produce una salida de error que es función de la posición actual y de la posición esperada. En cada iteración, el algoritmo modifica la posición del individuo utilizando un vector de velocidad $\overrightarrow{v}^i$ asociado a la partícula:

$$\overrightarrow{x}^i(t) = \overrightarrow{x}^i(t-1) + \overrightarrow{v}^i(t-1) \tag{6.2}$$

Algoritmo 6.8 Optimización por enjambre de partículas

```
{x_best} =PSO(N, V, D: integer; f: TipoFuncionObjetivo)
var
  /*N es el número de partículas*/
  /*V es el número de vecindades*/
  /*D es el número de dimensiones*/
  p, x: array [1...N] of TipoSolucion;
  v: array [1...N] of TipoVecindad; // Población de vecindades
  x_best: TipoSolucion;

  begin
    g := 0; // Inicializar el número de generación
    {p^i} := InicializarParticulas(N); // Inicializar los elementos de la población
    repeat
      for i := 1 to N do
        {p[i]} := ActualizarMejorParticula(p[i], f); // Almacenar el mejor valor
        de cada partícula
        for n := 1 to V do
          {v[n]}:=ActualizarMejorParticula(p[n], p[i], f); // Almacenar el mejor
          valor de cada vecindad
        end for
        g := g + 1
        for d := 1 to D do
          v^i_g = v^i_{g-1} + ρ1(p^i - v^i_{g-1}) + ρ2(p^n - v^i_{g-1}); // Actualizar la velo-
          cidad
          x^i_g = x^i_{g-1} + ρ1(v^i_g; // Actualizar la posición
        end for
      end for
    until terminacion
    x_best := SeleccionarMejor(p, f); // Seleccionar la mejor solución encontrada
  end
```

Las partículas del enjambre producen una señal de error en cada iteración con el objetivo de modificar su velocidad, cambiando de posición y describiendo una trayectoria en el espacio de soluciones. Si $\vec{p}^{\,i}$ es la mejor posición que ha encontrado en esa trayectoria, la distancia $\vec{d}^{\,i}$ entre la posición actual $\vec{x}^{\,i}$ y la mejor posición encontrada viene dada por:

$$\vec{d}^{\,i} = \vec{p}^{\,i} - \vec{x}^{\,i} \tag{6.3}$$

que determina la dirección y la distancia que la partícula debería tener para regresar al mejor valor conocido. El algoritmo de enjambre de partículas utiliza esta diferencia para establecer el valor del vector de velocidad. Su descripción matemática se puede expresar mediante la siguiente ecuación:

$$\vec{v}^{\,i}(t) = \vec{v}^{\,i}(t-1) + \rho_1(\vec{p}^{\,i} - \vec{x}^{\,i}(t-1)) \tag{6.4}$$

donde ρ_1 es un número aleatorio cuyo límite superior está prefijado (habitualmente $\rho_{1_{max}} = 2$).

La ecuación (6.4) describe una tendencia a volver hacia la mejor solución conocida. Este hecho se interpreta como el deseo de los organismos (partículas) por repetir comportamientos pasados o volver a situaciones exitosas (algo parecido a la "nostalgia") [177][176].

La interacción social se modela por medio de la definición de vecindad, de modo que cada partícula sólo interacciona con sus vecinas. Existen diferentes definiciones de vecindad [177][176] que se pueden utilizar en el contexto de SI. Aunque todos los individuos del enjambre realizan la búsqueda simultáneamente, a cada uno solamente le afecta su vecindad, de modo que los individuos de esa vecindad contribuyen en la búsqueda de un mejor valor para las coordenadas. Si $\vec{p^g}$ es la mejor posición que se ha encontrado en esa vecindad, la dirección y distancia $\vec{d^g}$ entre la posición actual $\vec{x^i}$ de cada individuo de esa vecindad y la mejor posición encontrada viene dada por:

$$\vec{d}^{\,g} = \vec{p}^{\,g} - \vec{x}^{\,i} \tag{6.5}$$

Como en el caso anterior, este valor también se aleatoriza para modificar la velocidad. En [177][176] se interpreta como la tendencia que tienen los individuos a imitar los éxitos de los demás. Finalmente, el vector velocidad, considerando los dos efectos descritos anteriormente se puede calcular como:

$$\vec{v}^{\,i}(t) = \vec{v}^{\,i}(t-1) + \rho_1(\vec{p}^{\,i} - \vec{v}^{\,i}(t-1)) + \rho_2(\vec{p}^{\,g} - \vec{v}^{\,i}(t-1)) \tag{6.6}$$