Metaheurísticas

Daniel A. Neira

Octubre, 2018

En este espacio revisaremos algunas de las metaheurísticas que se encuentran en las dos grandes familias de metaherísticas las S-Metaheurísticas, basados en la búsqueda una solución, y las P-Metaheurísticas, basadas en búsquedas sobre poblaciones[1].

Índice

List of Algorithms

Índice de cuadros

[1]Traducido del libro Metaheuristics: From design to Implementation [Talbi, 2009]

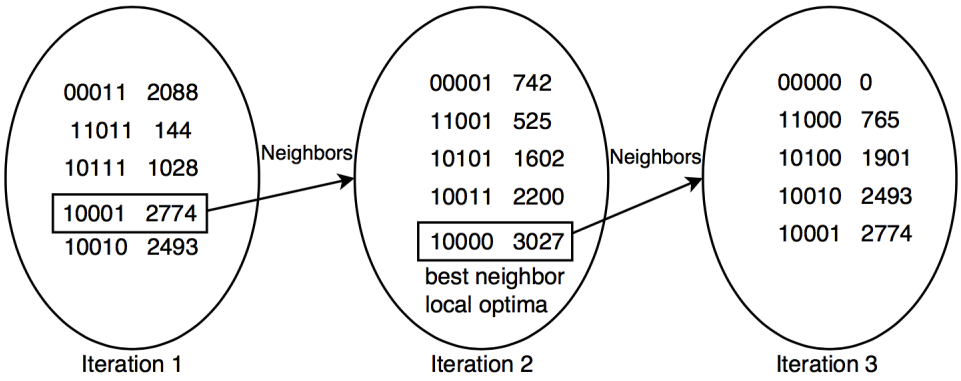

Figura 1: Local Search, en este ejemplo, utiliza una representación binaria de las soluciones, un operador de *flip move*, y una estrategía de selección del mejor vecino. La función objetivo es maximizar x^3-60x^2+900x.La solución óptima global es $f(01010) = f(10) = 4000$, mientras que el último óptimo local encontrado es $s = (1000)$, comenzando con una solución $s_0 = (10001)$.

1. Metaheurísticas basadas en una solución (S-Metaheurísticas)

Cuando se está resolviendo problemas de optimización, las metaheurísticas basadas en una solución o S-Metaheurísticas mejoran una solución dada. Pueden ser vistos como caminatas sobre vecindarios o búsqueda de trayectorias a través del espacio de búsqueda del problema que se tiene. Estas trayectorias son realizadas por procedimientos iterativos que se mueven desde la solución actual a otra en el espacio de búsqueda. S-Metaheurísticas muestran su eficiencia al resolver varios problemas de optimización en diferentes dominios. En este capitulo, una vista unificada de los conceptos comunes de búsqueda para esta clase de metaheurísticas es presentado. Luego, los principales algoritmos existentes, tanto desde el punto de vista del diseño como la implementación de forma incremental.

1.1. Local Search (LS)

Local Search o búsqueda local[2] es probablemente el más antiguo y simple método metaheurístico [Aarts et al., 2003]. Inicia con una solución inicial dada. En cada iteración, la metaheurística reemplaza la solución actual por medio de un vecindario que mejora la función objetivo (ver Figura 1). La búsqueda se detiene cuando todos los posibles vecindarios (o vecindarios candidatos) son peores que la solución actual, es decir, se ha obtenido un óptimo local. Para vecindarios de gran magnitud, la solución candidata puede ser un subconjunto de los vecindarios. El objetivo principal de esta estrategia de vecindario restringido es de acelerar la búsqueda. Variantes de LS pueden ser distinguidos de acuerdo con el orden en el cual las soluciones vecinas son generadas (determinísticas/estocásticas) y la estrategia seleccionada (selección de las soluciones vecinas) Figura 2. LS puede ser visto como un camino de descenso en el grafo que representa el espacio de búsqueda. Este grafo puede definirse por $G = (S, V)$ donde S representa el conjunto de todas las posibles soluciones del espacio de búsqueda y V representa la relación de vecindad. En el grafo G, los nodos (i, j) conectarán a toda solución vecina s_i y s_j. Para una solución s dada, el numero de nodos asociados será $|N(s)|$ (número de vecindarios). El Algoritmo 1 ilustra el modelo del algoritmo de búsqueda local.

Desde una solución inicial s_0, el algoritmo generara una secuencia $s_1, s_2, ..., s_k$ de soluciones con las siguientes características:

- El tamaño de la secuencia k es desconocido a priori
- $s_{i+1} \in N(s_i),\ \forall i \in [0, k-1]$
- $f(s_{i+1}) < f(s_i),\ \forall i \in [0, k-1]$[3]

[2]También llamado ascenso de colina, descenso, mejora iterativa, entre otros.
[3]Suponiendo que estamos frente a un problema de minimización.

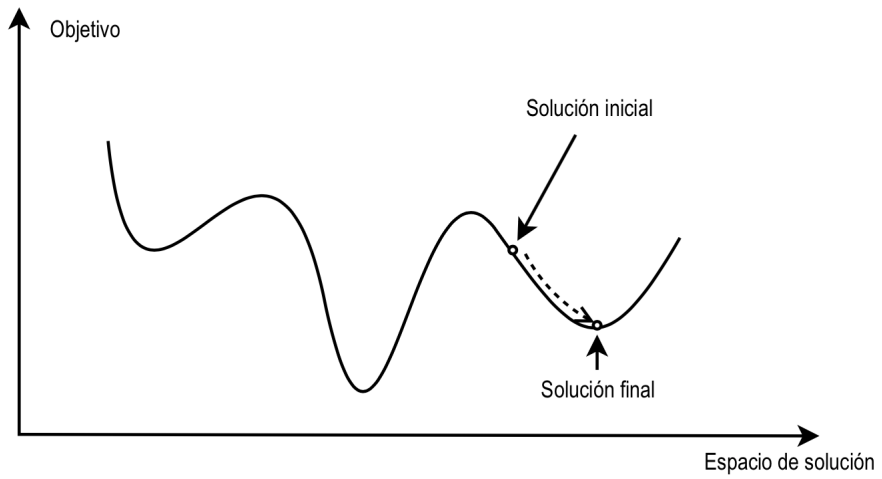

Figura 2: Local Search comportamiento (más empinado) en un panorama dado.

Algorithm 1 Algoritmo de búsqueda local.

$s = s_0$; /*Generar una solución inicial s_0*/
while no Criterio_Termino **do**
  Generar $N(s)$; /*Generación de vecindarios candidatos*/
  if no hay mejor vecindario **then**
    break
  end if
  $s = s^{'}$; /*Seleccionar el mejor vecindario $s^{'} \in N(s)$*/
end while
Output Solución final encontrada (óptimo local)

- s_k es un óptimo local: $f(s_k) \leq f(s),\ \forall s \in N(s_k)$

Adicionalmente a la definición de la solución inicial y el vecindario, diseñar un algoritmo de búsqueda local tiene que tomar en consideración la estrategia de selección del siguiente vecindario a visitar de manera cuidadosa.

1.2. Simulated Annealing (SA)

Simulated annealing o recocido simulado aplicado para problemas de optimización emerge en el trabajo de S. Kirkpatrick et al. [Kirkpatrick et al., 1983] y V. Cerni [Černỳ, 1985]. En estos trabajos pioneros, SA ha sido aplicado a partición de grafos y diseño de VLSI (En español, integración a muy gran escala). En los 1980's, SA tuvo un gran impacto en el campo la búsqueda de heurísticas por su simplicidad y eficiencia en resolver problemas de optimización combinatoria. Desde entonces, ha sido extendido para tratar con problemas de optimización continua. SA esta basado en los principios de la mecánica estadística en donde el proceso de recosido requiere calentar y luego enfriar lentamente una sustancia para obtener una estructura cristalina fuerte. La fuerza de la estructura depende de la tasa de enfriamiento de los metales. Si la temperatura inicial no es suficientemente alta o un enfriamiento rápido es aplicado, imperfecciones (estados metaestables) se obtienen. En este caso, el sólido refrigerante no alcanzará un equilibrio termal a esa temperatura. Cristales fuertes crecen debido a un cuidadoso y lento enfriamiento. EL algoritmo SA simula los cambios de energía en un sistema subyacente a un proceso de enfriamiento hasta su convergencia a un estado de equilibrio (estado de congelación constante). Este esquema fue desarrollado en 1953 por Metropolis [Metropolis et al., 1953]. El Cuadro 1 ilustra la analogía entre sistemas físico y el problemas de optimización. La función objetivo del problema es análoga al estado de la energía del sistema. Una solución del problema de optimización corresponde a un estado del sistema. Las variables de decisión asociadas con una solución del problema

Cuadro 1: Analogía entre sistemas físicos y problemas de optimización.

Sistema Físico	Problema de Optimización
Estado del Sistema	Solución
Posiciones moleculares	Variables de decisión
Energía	Función objetivo
Estado fundamental	Solución optima global
Estado metaestable	Óptimo local
Enfriamiento rápido	Búsqueda local
Temperatura	Parámetro de control T
Recocido cuidadoso	*Simulated Annealing*

son análogas a la posición de las moléculas. El óptimo global corresponde con el estado fundamental del sistema. Encontrar un mínimo local implica que un estado metaestable fue encontrado. SA es un algoritmo estocástico que permita bajo ciertas condiciones la degradación de una solución. El objetivo es escapar de un óptimo local y así retrasar la convergencia. SA es un algoritmo sin memoria en el sentido que el algoritmo no usa ninguna información reunida durante la búsqueda. Desde una solución inicial, SA procede en variadas iteraciones. En cada iteración, un vecino aleatorio es generado. Movimientos que mejoran la función de costos son siempre aceptados. De otra forma, el vecino es seleccionado con una probabilidad dada que depende de la temperatura actual y la cantidad de degradación ΔE de la función objetivo. ΔE representa la diferencia en la función objetivo (energía) entre la solución actual y la solución vecina generada (Ver Figura 3). Dado como el algoritmo progrese, la probabilidad que estos movimientos sean aceptados disminuye. Esta probabilidad sigue, en general, la distribución de Boltzmann:

$$P(\Delta E, T) = e^{-\frac{f(s^{'})-f(s)}{T}} \tag{1}$$

Se usa un parámetro de control, llamado temperatura, que determina la probabilidad de aceptar una solución sin mejora. En un nivel particular de temperatura, muchos tentativas son exploradas. Una vez que el equilibrio es alcanzado la temperatura decrece gradualmente de acuerdo con un programa de enfriamiento tal que pocas soluciones sin mejorar sean aceptadas al final de la búsqueda. El Algoritmo 2 describe el modelo para el Algoritmo SA.

En adición con la solución actual, la mejor solución encontrada desde el comienzo de la búsqueda es almacenada. Pocos parámetros controlan el progreso de la búsqueda, los cuales son la temperatura y el número de iteraciones ejecutadas en cada temperatura.

1.3. Tabu Search (TS)

Tabu search o búsqueda tabú es un algoritmo propuesto por Glover [Glover, 1989]. En 1986, él tomo la idea de la aleatorización controlada en SA para escapar de los óptimos locales y propuso un algoritmo determinístico. En un trabajo paralelo un acercamiento similar llamado "Ascenso más empinado/ Descenso más suave", fue propuesto por Hansen [Hansen, 1986]. En los 1900's, el algoritmo de tabu search se convirtió en muy popular en resolver problemas de optimización de una forma aproximada. Hoy en día, es una de las S-Metaheuristicas más utilizadas. El uso de memoria, que almacena información relacionada con la búsqueda del proceso, representa la característica particular de la búsqueda tabú. TS se comporta como un algoritmo steepest LS, pero éste acepta soluciones sin mejora para escapar de los óptimos locales cuando todos los vecindarios no tienen mejores soluciones. Usualmente, todo el vecindario es explorado en una manera determinística, en donde en SA un vecino es aleatoriamente seleccionado. Cuando un óptimo local es alcanzado, la búsqueda sigue adelante seleccionando a un peor candidato que la solución actual. La mejor solución en el vecindario es seleccionada como la nueva solución actual, incluso si esta no mejora la solución actual. Tabu Search puede ser visto como una transformación dinámica de el vecindario. Esta política puede generar ciclos en la que soluciones previamente visitadas pueden ser seleccionada nuevamente. Para evitar ciclos, TS descarta los vecinos que ya han sido visitados previamente. Se memoriza las trayectorias de búsqueda reciente. Tabu search maneja una memoria de soluciones o movimientos recientemente aplicados,

Algorithm 2 Algoritmo Simulated Annealing.

Input: Programa de enfriamiento
$s = s_0$; /*Generar una solución inicial s_0*/
$T = T_{max}$ /*Temperatura inicial*/
repeat
  repeat
    /*A una temperatura fija*/
    Generar un vecino de forma aleatoria $s^{'}$
    $\Delta E = f(s^{'}) - f(s)$;
    if $\Delta E \leq 0$ **then**
      $s = s^{'}$; /*Aceptar la solución vecina*/
    else
      Aceptar $s^{'}$ con una probabilidad $e^{-\frac{\Delta E}{T}}$;
    end if
  until Condición_Equilibrio
  /*Por ejemplo, un número dado de iteraciones ejecutadas para cada temperatura T */
  $T = g(T)$; /* Actualización de temperatura */
until Criterio_detención /* Por ejemplo, $T < T_{min}$ */
Output: Mejor Solución encontrada

la cual es llamada lista tabú. Esta lista tabú constituye la memoria de corto plazo. En cada iteración de TS, la memoria de corto plazo es actualizada. Almacenar todas las soluciones visitadas consume tiempo y espacio de memoria. De hecho, tenemos que chequear en cada iteración si una solución generada no pertenece a la lista de las soluciones visitadas. La lista tabu usualmente contiene un número constante de movimientos tabu. Usualmente, los atributos de los movimientos son almacenados en la lista tabú. Al introducir los conceptos de características de solución o movimiento en la lista tabú, uno puede perder alguna información acerca de la memoria de búsqueda. Podemos rechazar soluciones que no han sido generadas aun. Si un movimiento es "bueno" pero es tabú, ¿aún lo rechazamos? La lista tabú puede ser muy restrictiva; una solución no generada puede ser prohibida. Ahora para algunas condiciones, llamadas "criterios de aspiración", soluciones tabu pueden ser aceptadas. Las soluciones admisibles del vecindario son aquellas que no son tabu o pertenece/estar en los criterios de aspiración. Adicionalmente al diseño común de las S-metaheurísticas como la definición de vecindario y la generación de la solución inicial, los asuntos más importantes de diseño que son específicos para un TS simple son:

- Lista Tabú: El objetivo de usar una memoria de corto plazo es prevenir la búsqueda de soluciones previamente visitadas. Como se menciono, almacenando la lista de todas las soluciones visitadas no es practico por temas de eficiencia.

- Criterio de Aspiración: Un criterio de aspiración comúnmente usado consiste en seleccionar un movimiento tabú si este genera una solución que es mejor que la mejor solución encontrada hasta ahora. Otro criterio puede ser un movimiento tabú que proporciona una mejor solución entre un conjunto de soluciones que poseen un atributo dado.

Algunos avances mecánicos son comúnmente introducidos en la búsqueda tabú para lidiar con la intensificación de la búsqueda:

- Intensificación (memoria de mediano plazo): La memoria de mediano plazo almacena la elite (las mejores) soluciones encontradas durante la búsqueda. La idea es dar prioridad a los atributos de los conjuntos de solución elite, usualmente con probabilidades ponderadas. La búsqueda está sesgada por estos atributos.

- Diversificación (memoria de largo plazo): La memoria de largo plazo almacena la información de la visita de las soluciones durante la búsqueda. Explora las áreas no visitadas en el espacio de solución. Por ejemplo, desincentivará los atributos de las mejores soluciones en las soluciones generadas para diversificar la búsqueda a otras áreas en el espacio de búsqueda.

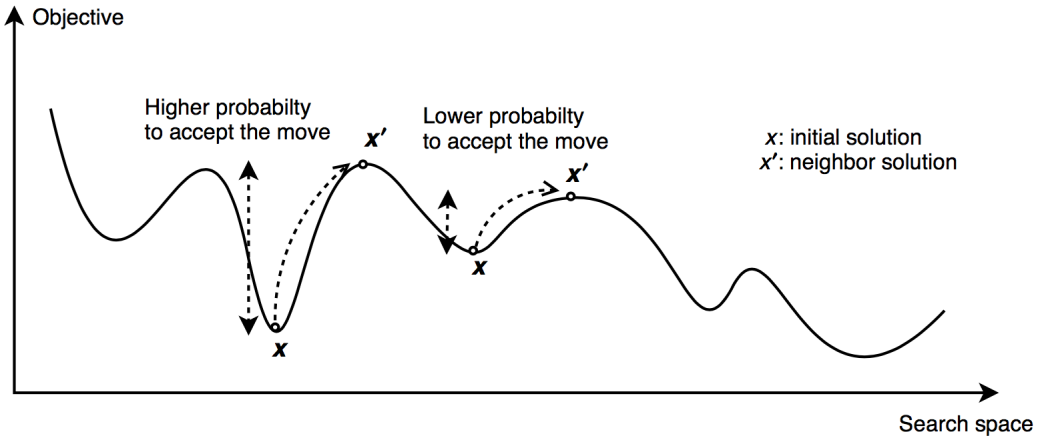

Figura 3: SA escapando de un óptimo local. Mientras más grande sea la temperatura más grande es la probabilidad de aceptar una movimiento subóptima. Dada una temperatura, mientras menor el incremento de la función objetivo, más grande será la probabilidad de aceptar un movimiento. Un movimiento mejor siempre es aceptado.

El Algoritmo 3 describe el modelo para el algoritmo de búsqueda tabú.

Algorithm 3 Algoritmo Tabu Search.

$s = s_0$; /* Solución inicial s_0*/
Inicializar lista tabu, memoria de mediano y largo plazo;
repeat
  Encontrar el mejor vecino admisible $s^{'}$; /* No tabu o con un criterio de aspiración */
  $s = s^{'}$;
  Actualizar lista tabu, condiciones de aspiración, memoria de corta y largo plazo;
  if criterio_intensificación **then**
    intensificación;
  end if
  if criterio_diversificación **then**
    diversificación;
  end if
until Criterio de detención
Output: Mejor Solución encontrada

Estudios teóricos llevados a cabo sobre el algoritmo tabu search son débiles que aquellos establecidos para simulated annealing. Una ejecución de simulated annealing se encuentra dentro del casco convexo del conjunto de las ejecuciones de tabu search, Althofer y Koschnick [Althöfer and Koschnick, 1991]. Por lo tanto, tabu search puede heredar algunas buenas propiedades de SA. Adicionalmente, en la búsqueda de componentes en la búsqueda local (hill climbling), tanto como la representación, vecindario, solución inicial, tenemos que definir los siguientes conceptos que componen la memoria de búsqueda de TS: la lista tabú (memoria de corto plazo), la memoria de mediano plazo y la de largo plazo (Ver Cuadro 2).

1.4. Iterated Local Search (ILS)

La calidad del optimo local obtenido vía método de búsqueda local depende de la solución inicial. Como los óptimos locales se generan con gran variabilidad, la búsqueda local iterada puede usarse para mejorar la calidad de los óptimos locales sucesivos. Este tipo de estrategias ha sido aplicada primero en [Martin et al., 1991] y luego generalizado en [Stützle, 1999]. En la búsqueda local multicomienzo (*multistart local search*), la solución inicial es siempre seleccionada aleatoriamente, y por ende, no está relacionada el comienzo con generar

Cuadro 2: Diferentes memorias de búsquedas del Tabu Search.

Memoria de búsqueda	Rol	Representación popular
Lista tabú	Prevenir ciclos	Soluciones visitadas, atributos de movimiento y solución
Memoria de mediano plazo	Intensificación	Memoria reciente
Memoria de largo plazo	Diversificación	Memoria de frecuencia

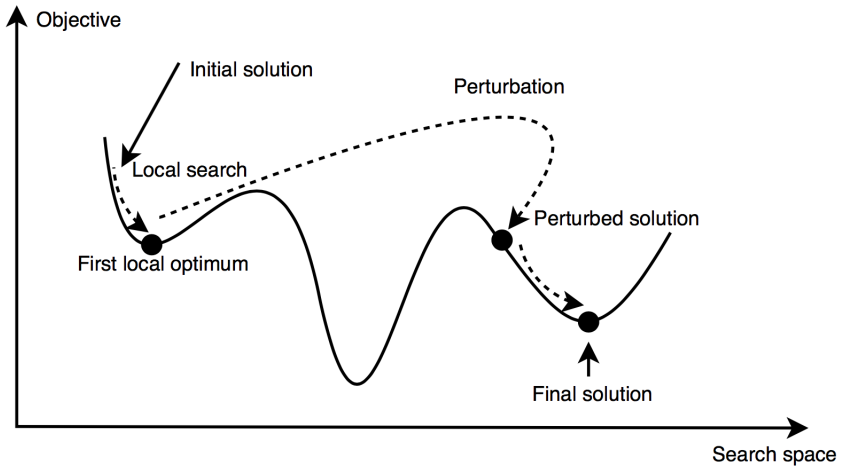

Figura 4: El principio del algoritmo de búsqueda local iterada (ILS).

el optimo local. **ILS** mejora la búsqueda local multicomienzo clásica, "perturbando" el optimo local y reconsiderándolos como soluciones iniciales. El fenómeno del limite central en el paisaje de un problema de optimización implica que cuando el tamaño de un problema se convierte en muy grande, el optimo local obtenido usando diferentes soluciones iniciales son mas o menos similares en términos de calidad ([Baum, 1986]). Por tanto, los multi-comienzos generalmente fallan para problemas con instancias muy grandes ([Schoen, 1991]). **ILS** esta basado en un principio simple que ha sido usado en muchas heurísticas especificas como la heurística iterada Lin-Kerningham para el problema del vendedor viajero ([Johnson, 1990]) y la búsqueda tabú adaptativa para el problema de asignación cuadrático ([Talbi et al., 1998]). Primero, una búsqueda local es aplicada para una solución inicial. Luego, en cada iteración, una perturbación de el optimo local obtenido es efectuada. Finalmente, una búsqueda local es aplicada para tal solución perturbada. La solución generada es aceptada como la nueva solución actual bajo algunas condiciones. Este proceso itera hasta un criterio de detención dado. El Algoritmo 4 describe ILS.

Algorithm 4 Algoritmo de Búsqueda Local Iterada.

$s_* = localsearch(s_0)$; /* Aplicar un algoritmo de búsqueda local dado */
repeat
  $s^{'} = Perturbar(s_*, historiadebusqueda)$; /* Perturbar el optimo local obtenido*/
  $s^{'} = localsearch(s^{'})$; /* Aplicar búsqueda local sobre la solución perturbada */
until Criterio de detención
Output: Mejor solución encontrada

Tres elementos básicos componen un ILS (Ver Figura 4).

- **Local Search:** Cualquier S-Metaheurística (determinística o estocástica) puede ser usada en la estructura de ILS tal como un algoritmo descendente simple, un tabu search, o simulated annealing. El proceso de búsqueda es tratado como una caja negra (Ver Figura 5). En la literatura, las P-Metaheurísticas no son consideradas como candidatas en el proceso de búsqueda debido a que manipulan las poblaciones, sin embargo, algunas metaheurísticas basadas en poblaciones integran el concepto de perturbación de

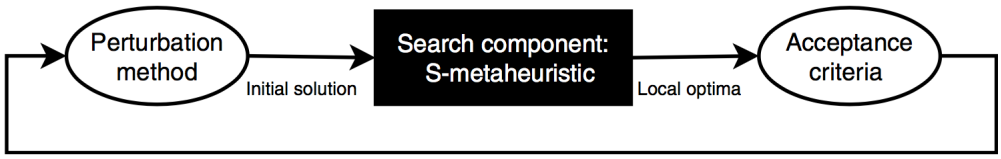

Figura 5: El componente de búsqueda es visto como una caja negra para algoritmo ILS.

la (sub)población para estimular la diversificación de la búsqueda.

- **Método de perturbación:** El operador perturbador puede ser visto como un gran movimiento aleatorio de la solución actual. El método de perturbación debe mantener una parte de la solución y perturbar fuertemente otra parte de la solución para mover, con suerte, a otra cuenca de atracción.
- **Criterio de aceptación:** El criterio de aceptación define las condiciones que el nuevo optimo local debe satisfacer para reemplazar la solución actual.

Una vez que la S-Metaheurística envuelta en la estructura del **ILS** está especificada, el diseño de ILS dependerá principalmente en el uso del método de perturbación y el criterio de aceptación. Muchos diseños diferentes pueden ser definidos de acuerdo a varias decisiones para implementar el método de perturbación y el criterio de aceptación.

1.4.1. Método de Perturbación

La primera motivación del algoritmo ILS se basa en el hecho de que el método de perturbación debe ser más eficaz que un enfoque de reinicio aleatorio, en el que una solución inicial independiente se regenera de forma aleatoria. Esto ocurrirá para la mayoría de los problemas de optimización de la vida real que están representados por paisajes de búsqueda estructurados. Para esos problemas, se debe llevar a cabo una perturbación sesgada. Sin embargo, para algunos problemas e instancias, donde el paisaje es accidentado y plano, el reinicio aleatorio puede ser más eficiente. Un primer ejercicio en el diseño de un algoritmo de ILS para resolver el problema en cuestión será comparar el desempeño del reinicio aleatorio y la perturbación sesgada implementada. Se debe encontrar un buen equilibrio para la perturbación sesgada. La longitud de una turbulencia puede estar relacionada con el vecindario asociado con la codificación o con el número de componentes modificados. Una perturbación demasiado pequeña puede generar ciclos en la búsqueda y no se obtiene ninguna ganancia. La probabilidad de explorar otras cuencas de atracción será baja. Además, aplicar una búsqueda local a una pequeña perturbación es más rápido que para grandes perturbaciones. Una perturbación demasiado grande borrará la información sobre la memoria de búsqueda y, a continuación, se omitirán las buenas propiedades de las óptimas locales. El extremo de esta estrategia es el enfoque de reinicio aleatorio. Cuanto más efectivo es la S-Metaheurística, mayores deben ser los valores de la perturbación. La longitud óptima depende principalmente de la estructura del paisaje del problema de optimización y no debe exceder la longitud de correlación. Incluso para un problema específico, dependerá de la instancia en cuestión. El operador de movimiento utilizado en la perturbación puede ser de naturaleza diferente de la relación de vecindad utilizada en el procedimiento de búsqueda local. Muchos métodos de perturbación sesgada pueden diseñarse de acuerdo con los siguientes criterios:

- Perturbaciones fijas o variables. La duración de las perturbaciones aplicadas a una Óptima local puede definirse de la siguiente manera:
  - Estática. La longitud se fija *a priori* antes del comienzo de la búsqueda de ILS.
  - Dinámico. La longitud de la perturbación se define dinámicamente sin tomar en cuenta en cuenta la memoria de búsqueda.
  - Adaptable. En esta estrategia, la duración de la perturbación se adapta durante la búsqueda según algunas informaciones sobre la memoria de búsqueda. De hecho, la longitud óptima dependerá de la instancia de entrada y de su estructura. Más información sobre las características del paisaje puede ser extraída durante la búsqueda.

- Perturbación aleatoria o semideterminística. La perturbación que se produce en una solución puede ser aleatoria (sin memoria), en la que cada movimiento se genera de forma aleatoria en el barrio. Esto conduce a una cadena Markoviana. En las perturbaciones semipermanentes-terminista, el movimiento está sesgado de acuerdo con la memoria de la búsqueda. Por ejemplo, se pueden aplicar las tareas de intensificación y diversificación del algoritmo de búsqueda tabú utilizando la memoria a medio plazo y la memoria a largo plazo. El primer enfoque es más popular en la literatura, mientras que el segundo necesita mecanismos de búsqueda más avanzados y complejos para ser implementados.

1.4.2. Criterios de aceptación

El papel del criterio de aceptación combinado con el método de perturbación es controlar el equilibrio clásico entre las tareas de intensificación y las de diversificación. La primera solución extrema en términos de intensificación es aceptar sólo soluciones de mejora en términos de la función objetiva (selección fuerte). La solución extrema en términos de diversificación es aceptar cualquier solución sin tener en cuenta su calidad (selección débil). Se pueden aplicar muchos criterios de aceptación que equilibran los dos objetivos:

- Criterios de aceptación probabilística: En la literatura se pueden encontrar muchos criterios de aceptación probabilística diferentes. Por ejemplo, la distribución de Boltzmann de recocido simulado. En este caso, se debe definir un programa de enfriamiento.

- Criterios de aceptación determinística: Algunos criterios de aceptación determinista pueden inspirarse en los algoritmos de aceptación de umbral y en los algoritmos relacionados, como el gran diluvio y los algoritmos de registro a registro.

2. Metaheurísticas basadas en población (P-Metaheurísticas)

Las metaheurísticas basas en población (P-Metaheurísticas) comparte muchos conceptos en común. Deben ser vistas como una mejora iterativa en una población de soluciones. Primero, la población se inicializa. Luego, una nueva población de soluciones es generada. Finalmente, esta nueva población es integrada dentro de la actual usando algún proceso de seleeción. El proceso de selección es detenedo cuando una condición dada es satisfecha (Criterio de detención). Algoritmos como algoritmos evolutivos (EAs), scatter search (SS), algoritmos de estimación de distribución (EDAs), optimización de enjambre de partículas (AISs) y sistemas inmunes artificiales (AISs) pertenecen a esta clase de metaheurísticas.

2.1. Algoritmos Evolutivos (EA)

En el siglo 19, J. Mandel fue el primero en indicar líneas base de la herencia de los padres a los hijos. Luego en 1859, C. Darwin presento la teoría de la evolución en su famoso libro El origen de las Especies [Charles, 1859]. En los 1980's, estas teorías de creación de nuevas especies y su evolución han inspirado científicos computacionales en diseñar algoritmos evolutivos. Diferentes escuelas principales de algoritmos genéticos han evolucionado independientemente en los últimos 40 años: algoritmos genéticos (GA), principalmente desarrollados en Michigan, EE.UU, por J. Holland ([Holland, 1962] y [Holland, 1992]); Estrategias evolucionarias, desarrolladas en Berlín, Alemania, por I. Rechenberg ([Rechenberg, 1965] y [Rechenberg,]) and H-P. Schwefel ([Schott, 1995]); y programación evolutiva por L. Fogel en San Diego, EE.UU ([Fogel, 1962] y [Fogel et al., 1966]). Más tarde, en los últimos años de los 1980's, la programación genética fue propuesta por J. Koza [Koza, 1994]. Cada uno de estos constituye un enfoque diferente; sin embargo, todas están inspiradas por los mismos principios evolutivos de la naturaleza. Los algoritmos evolucionarios son P-Metaheuristicas estocásticas que han sido satisfactoriamente aplicadas a muchos problemas de la vida real y problemas complejos (epistático, multimodal, multi-objetivo, y con problemas con muchas restricciones). Son los algoritmos basados en población más estudiados. Su éxito en resolver problemas difíciles de optimización en varios dominios (optimización continuo o combinatorial, modelación e identificación de sistemas, planificación y control, diseño de ingeniería, data mining y machine learning, vida artificial,etc) ha promovido el campo conocido como computación evolucionaria (EC).

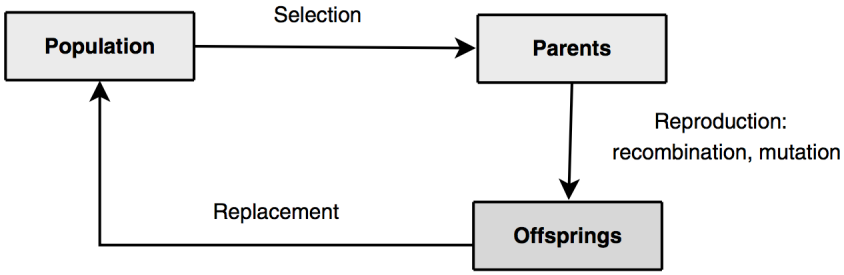

Figura 6: Una generación de algoritmo evolutivo.

Cuadro 3: Procesos evolutivos versus problemas de optimización.

Metáfora	Optimización
Evolución	Resolución de problemas
Individuo	Solución
Fitness	Función objetivo
Ambiente	Problema de optimización
Lugar	Elemento de la solución
Alelo	Valor de los elementos (localización)

EAs están basados en la noción de competición (ver Figura 6). Representan una clase de algoritmos iterativos de optimización que simulan la evolución de especies. Están basados en la evolución de una población de individuos. Inicialmente, esta población es usualmente generada aleatoriamente. Cada individuo en la población es la versión codificada de una solución tentativa. Una función objetivo asocia un valor fitness con cada individuo indicando su conveniencia para el problema. Entonces, los individuos seleccionados son reproducidos usando varios operadores (por ejemplo, cruzamiento, mutación) para generar nuevas generaciones. Este proceso es iterado hasta alcanzar un criterio de detención. El Cuadro 3 hace el paralelo entre la metáfora del proceso de evolución y resolver un problema de optimización. El Algoritmo 5 muestra el modelo de un algoritmo evolucionario.

Algorithm 5 Algoritmo Evolutivo.

Generar($P(0)$); /* Solución inicial */
$t = 0$;
while no Criterio_Termino($P(t)$) **do**
  Evaluar($P(t)$);
  $P^{'}(t)$ =Selección($P(t)$);
  $P^{'}(t)$ =Reproducción($P(t)$);
  Evaluar($P(t)$);
  $P(t+1)$ =Reemplazar($P(t), P^{'}(t)$);
  $t = t + 1$;
end while
Output: Mejor individuo o meor población encontrada.

En los algoritmos evolutivos, el genotipo representa la codificación mientras que el fenotipo representa la solución. Por esto, el genotipo debe ser decodificado para generar el fenotipo. Los distintos operadores actúan sobre el nivel de genotipo mientras que la función Fitness usará el fenotipo del individuo asociado. El Fitness de un individuo mide la habilidad del individuo a sobrevivir en su ambiente. En el caso donde una codificación directa es usada, el genotipo será similar al fenotipo. De otra forma (por ejemplo, si una codificación indirecta es usada), el genotipo y el fenotipo son estructuras distintas. De hecho, una función de codificación es usada para transformas el genotipo en un fenotipo (ver Figura 7). En términos de una

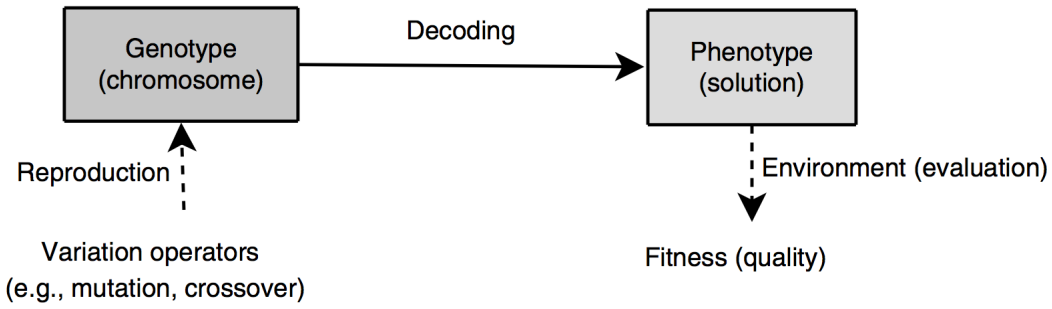

Figura 7: Genotipo versus fenotipo en algoritmos evolutivos.

convergencia asintótica teórica, muchos estudios muestran las propiedades de convergencia de los algoritmos evolutivos en los cuales los algoritmos son generalmente limitados a operadores de mutación y reemplazo elitista. En la sección siguiente, la principal diferencia entre las varias familias de los algoritmos evolutivos es esbozada. Por ende, el foco es hecho sobre sus componentes de búsqueda comunes.

2.2. Algoritmo Genéticos (GA)

Los algoritmos genéticos han sido desarrollados por J. Holland en los 1970's (Universidad de Michigan, EE.UU.) para entender el proceso adaptativo de sistemas naturales [Holland, 1992]. Fueron aplicados para problemas de optimización y machine learning en los 1980's. GA son una clase muy popular de EAs. Tradicionalmente, GAs son asociados con el uso de una representación binaria pero hoy en día uno puede encontrar GAs que usan otros tipo de representaciones. Un GA usualmente implica un operador de cruzamiento para dos soluciones que juega un rol mayor, además un operador de mutación que modifica aleatoriamente el contenido individual para promover diversidad (vea Tablas 4 y 5). GAs usan una selección probabilística que es originalmente la proporción seleccionada, esto es, los padres son remplazados sistemáticamente por su descendencia. El operador de cruzamiento está basado en los n-puntos o cruzamientos uniformes mientras la mutación es un operador de *swap*. Una probabilidad fija p_m (respectivamente p_c) es aplicado al operador de mutación (respectivamente de cruzamiento). A continuación se muestra la estructura del algoritmo genético en Algoritmo 6

Algorithm 6 Algoritmo Genético.

$t \leftarrow 0$
Inicializar $P(t)$
Evaluar $P(t)$
while No alcance condición de término **do**
  Inicio
  $t \leftarrow t + 1$
  Seleccinar $P(t)$ desde $P(t-1)$
  Alterar $P(t)$
  Evaluar $P(t)$
end while

2.3. Scatter Search (SS)

Scatter Search o búsqueda de dispersión tiene sus orígenes en el trabajo de F.Glover [Glover, 1977]. SS es una estrategia determinística que ha sido aplicada con éxito a algunos problemas continuos y de combinatoria. Incluso si los principios de los métodos han sido definidos desde 1977, las aplicaciones de SS están en sus comienzos. SS es una metaheuristica evolutiva y de población que recombina soluciones

Cuadro 4: Características principales de los diferentes alogitmos evolutivos: GA y ES.

Algoritmo	Algoritmo Genético	Estrategía Evolutiva
Desarrollador	J. Holland	I.Rechenberg, H.-P.Schwefel
Aplicación original	Optimización Discreta	Optimización Continua
Características de los atributos	No muy rápida	Optimización Continua
Características especiales	Cruzamiento, muchas variantes	Rápido, mucha teoría
Representación	Cadenas binarias	Vectores de valores reales
Recombinación	n-puntos o uniforme	Discreta o intermedia
Mutación	Volteo de bits con probabilidad fija	perturbación de Gaussian
Selección	Fitness	Uniforme
(selección de padres)	proporcional	Aleatorio
Reemplazo	Todos los hijos	(λ, μ)
(selección sobrevivientes)	reemplaza los padres	$(\lambda + \mu)$
Especialización	Enfásis en cruzamiento	Auto-adaptación
		del tamaño del paso de mutación

Cuadro 5: Características principales de los diferentes algoritmos evolutivos: EP y GP.

Algoritmo	Programación Evolutiva	Programación Genética
Desarrollador	D. Fogel	J. Koza
Aplicación original	Machine Learning	Machine Learning
Características de los atributos	-	Lento
Características especiales	No hay recombinación	-
Representación	Estados finitos de máquina	Árboles de análisis
Recombinación	No	Intercambio de sub-árboles
Mutación	perturbación gaussiana	Cambios aleatorios en árbol
Selección	Determinística	Proporcional al Fitness
Reemplazo	Probabilístico	
(selección sobrevivientes)	$(\mu + \mu)$	reemplazo generacional
Especialización	Auto-adaptación	Necesita gran población
		del tamaño del paso de mutación

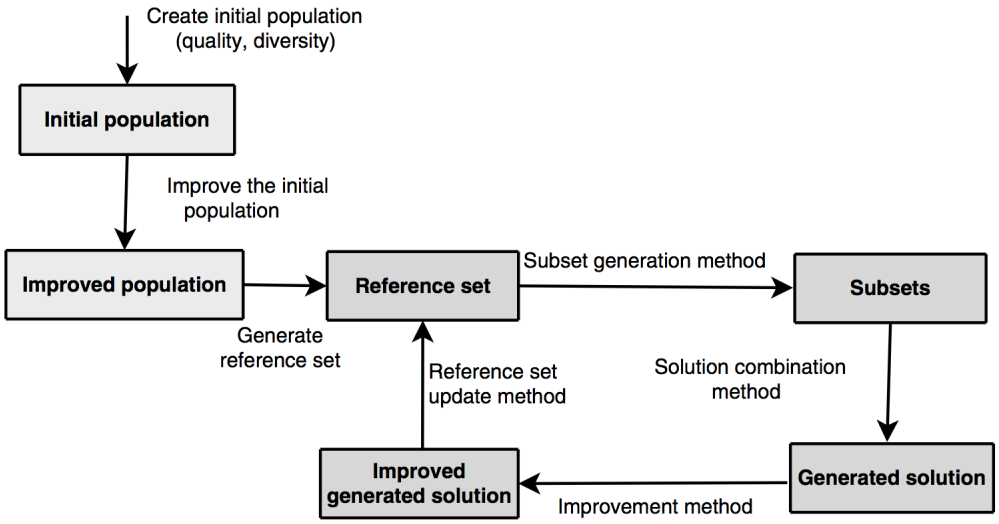

Figura 8: Componentes de búsqueda del algoritmo Scatter Search.

seleccionadas desde un conjunto de referencia para construir otros. El método comienza generando una población inicial Pop que satisface el criterio de diversidad y calidad. El conjunto de referencias ($RefSet$) de tamaño moderado (Típicamente, el conjunto de referencia está compuesta por más o menos 10 soluciones) es entonces construido seleccionando buenas soluciones representativa desde la población. Las soluciones seleccionadas son combinadas para proveer soluciones iniciales a un proceso de mejora basado en una S-Metaherística. De acuerdo con el resultado de tal proceso, el set de referencia e incluso la población de soluciones son actualizadas para incorporar una alta calidad y una diversidad de soluciones. El proceso es iterado hasta que un criterio de detención es satisfecho. El enfoque SS involucra diferentes procesos permitiendo generar la población inicial, para crear y actualizar el conjunto de referencia, para combinar las soluciones de tal conjunto, mejorar las soluciones construidas y así susesivamente. SS usa explícitamente estrategias para la intensificación y diversificación de la búsqueda. El algoritmo comienza con un conjunto de soluciones diversas, las cuales representan el conjunto inicial que se desarrolla por medio de la recombinación de soluciones así como por la aplicación de la búsqueda local (U otra S-metaheurística).

Algorithm 7 Algoritmo Scatter Search

```
/* Fase inicial */
Inicializar la población POP usando un método de generación de diversificación;
Aplicar el método de mejoramiento de la población;
Método de actualización del conjunto de referencia;
/* Iteración de Scatter Search */
repeat
  Método de generación de subconjuntos;
  repeat
    Método de combinación de soluciones;
    Método de mejora;
  until Criterio de detención 1
  Método de actualización del conjunto de referencia;
until Criterio de detención
Output: Mejor de solución encontrada o conjunto de soluciones.
```

El diseño de un algoritmo de Scatter Search esta generalmente basado en los siguientes cinco métodos (Figura 8).

- Método de generación de diversificación: Este método genera un conjunto de soluciones iniciales di-

versas. En general, procesos golosos son aplicados para diversificar la búsqueda mientras seleccionan soluciones de gran calidad.

- Método de mejora: Este método transforma una solución de prueba en una o más soluciones de prueba mejoradas usando cualquier S-Metaherística. En general, un algoritmo de búsqueda local es aplicado y entonces un óptimo local es generado.

- Método de actualización del conjunto de referencia: En este componente de búsqueda, un set de referencia es construido y mantenido. El objetivo es asegurar diversidad mientras se mantienen soluciones de buena calidad. Por ejemplo, uno puede seleccionar soluciones $RefSet1$ con la mejor solución objetivo y luego agregar las soluciones $RefSet2$ con la mejor diversidad ($RefSet = RefSet1 + RefSet2$).

- Método de generación de subconjuntos: Este método opera sobre los conjuntos de referencia $RefSet$, para producir un subconjunto de soluciones como una base para crear soluciones combinadas. Este método usualmente selecciona todos los subconjuntos de un tamaño fijo r (en general, $r = 2$). Este procedimiento es similar al mecanismo de selección en EAs. Es generalmente un operador estocástico. Además, el tamaño del conjunto de referencia en SS es mucho más pequeño que el tamaño de la población en EAs. Esto es porque más mecanismos de selección enumerativa se pueden aplicar.

- Método de combinación de soluciones: Un dado subconjunto de soluciones producidas por el método de generación de subconjuntos. En general, combinaciones de estructurada ponderada son usadas vía combinaciones lineales y redondeo generalizado a variables discretas. Este operador puede ser visto como una generalización de el operador de cruzamiento en EAs donde más que dos individuos son recombinados.

Los tres componentes de búsqueda (método de generación de diversificación, método de mejora y método de combinación) son específicos para un problema, donde los dos otros componentes de búsqueda (método de actualización del conjunto de referencia, método de generación de subconjuntos) son genéricos. Estos métodos se combinan, como se muestra en la Figura 8 y el Algoritmo 7.

2.4. Algoritmo de Colonia de hormigas (ACO)

La idea básica del algoritmo de colonia de hormigas (ACO) es imitar el comportamiento cooperativo de hormigas reales para resolver problemas de optimización. Las metaheurísticas ACO han sido propuestas por M. Dorigo [Dorigo et al., 1996]. Pueden ser vistas como sistemas multi-agente en el cual cada agente está inspirado por el comportamiento de una hormiga real. Tradicionalmente ACO ha sido aplicado a problemas de optimización combinatoria y han logrado un amplio éxito en resolver diferentes problemas (por ejemplo, scheduling, routing, asignaciones). El principal interés del comportamiento de las hormigas reales es que las simples hormigas usan comportamientos colectivos logrando tareas complejas tales como transporte de comida y encontrar el camino más corto hacia las fuentes de comida. Los algoritmos ACO imitan el principio por el cual usando mecanismos de comunicación muy simples, una colonia de hormigas logra encontrar el camino más corto entre dos puntos. Como puede ser ilustrado en un ejemplo, tengamos una colonia real de hormigas que quieren crear un camino desde su nido hasta la fuente de comida. Esto lo logran mediando una feromona, creando un rastro químico en el piso. La feromona es una sustancia olfativa y volátil. El rol de este camino es guiar a las otras hormigas en dirección del objetivo. Mientras mayor sea la cantidad de feromona en un camino en particular, mayor es la posibilidad que las hormigas seleccionen este camino. Para una hormiga da, el camino es escogido de acuerdo a la cantidad olida de feromona. Aún más, esta sustancia química tiene una acción decreciente sobre el tiempo (Proceso de evaporación) y la cantidad dejada por una hormiga depende de la cantidad de comida (proceso de refuerzo). Cuando estas hormigas se encuentran frente a un obstáculo, existe una misma probabilidad para cada hormiga de elegir el camino a la izquierda o a la derecha. Cuando el camino izquierda sea más corto que el derecho y este, entonces, necesito menos tiempo de viaje, las hormigas terminaran dejando un mayor nivel de feromona. Mientras más hormigas tomen el camino de la izquierda, mayor será la feromona sobre el camino. Por tanto, hay una aparición del camino más corto. Este hecho aumentar por la etapa de evaporación. Esta indirecta forma de cooperación es conocida como estigmergia. El Algoritmo 8 presenta el modelo para ACO. Primero, la información de la feromona

Algorithm 8 Algoritmo ACO.

```
Inicializar los caminos de feromona;
repeat
  for cada hormiga do
    Construcción de la solución usando el camino de feromona;
    Actualizar los caminos de feromona;
    Evaporación;
    Refuerzo;
  end for
until Criterio de detención
Output: Mejor solución encontrada o conjunto de soluciones
```

es inicializada. El algoritmo es principalmente compuesto por dos pasos iterativos: La construcción de la solución y la actualización de la feromona.

- Construcción de la solución: La construcción de las soluciones se hace de acuerdo a la regla de la probabilidad de los estados transitorios. Hormigas artificiales pueden ser consideradas como procesos estocásticas golosos que construyen una solución en una forma probabilística agregando componentes de solución a algunas parciales ya creadas, hasta que una solución completa es derivada. El problema de optimización objetivo puede ser visto como una decisión grafica (o construcción gráfica) en donde una hormiga construirá un camino. Usualmente, este proceso iterativo tomara en consideración:
  - Caminos de feromonas: De hecho, los caminos de feromonas, memorizan las características de "buenas" soluciones generadas, las cuales guiaran la construcción de nuevas soluciones por las hormigas. Los caminos de feromona cambian dinámicamente durante la búsqueda para reflejar el conocimiento adquirido. Esto representa la memoria de todo el proceso de búsqueda con hormigas.
  - Problema dependiente de la información de la heurística: Un problema de información específico da más pistas a las hormigas en sus decisiones para construir soluciones.
- Actualización de la feromona: La actualización de la feromona es seguida usando las soluciones generadas. Una regla de actualización global de la feromona es aplicada en dos fases:
  - Una fase de evaporación donde el camino de feromona decrece automáticamente. Cada valor de feromona se reduce en una proporción fija:

    $$\tau_{ij} = (1-\rho)\tau_{ij}, \quad \forall i,j \in [1,n] \tag{2}$$

    Donde $\rho \in]0,1]$ representa la tasa de reducción de la feromona. El objetivo de la evaporación es reducir para todas las hormigas una convergencia prematura hacia las "buenas" soluciones y entonces alentar la diversificación en el espacio de búsqueda.
  - Una fase de reforzamiento donde el camino de feromona es actualizado de acuerdo con las soluciones generadas. Tres diferentes estrategias pueden ser aplicadas:
    - Actualización de feromona paso a paso en línea: El camino de feromona τ_{ij} se actualiza para cada hormiga en cada paso de la construcción de soluciones.
    - Actualización de feromona retrasada en línea: La actualización de la feromona se aplica una vez una hormiga genere una solución completa. Por ejemplo, en el ACO, cada hormiga actualizará la información de la feromona con un valor que es proporcional a la calidad de la solución encontrada. Mientras mejor sea la solución encontrada, mas acumulada estará la feromona.
    - Actualización fuera de línea de la feromona: La actualización del tren de feromonas es aplicado una vez que todas las hormigas han generado una solución completa. Este es el enfoque más popular donde diferentes estrategias pueden ser usadas:

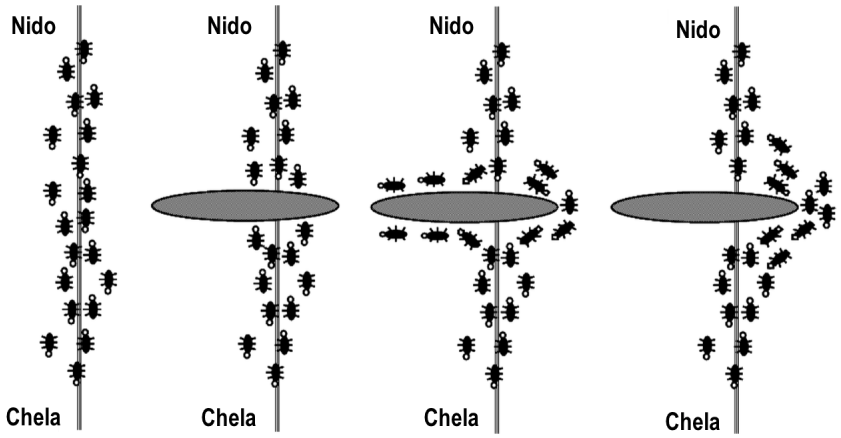

Figura 9: Fuente de inspiración de la búsqueda de colonia de hormigas para encontrar el camino óptimo entre la chela y su nido.

- ◇ Actualización de la feromona basada en la calidad: esta estrategia actualiza el valor de la feromona asociado con la mejor solución encontrada entre todas las hormigas (o las mejores k soluciones donde es un número menor al número de hormigas). El valor agregado depende de la calidad de las soluciones seleccionadas. Por ejemplo, para cada componente perteneciente a la mejor solución π^*, un valor positivo Δ es adherido:

$$\tau_{i\pi^*(i)} = \tau_{i\pi^*(i)} + \Delta, \quad \forall i \in [1, n] \tag{3}$$

- ◇ Actualización de la feromona basada en ranking: Las mejores soluciones de hormigas son permitidas a actualizar la feromona con una cantidad dependiente del ranking de las soluciones. Actualización demorada de la feromona: La hormiga generando la peor solución decrecerá su camino de feromona relacionado a los componentes de la solución.
- ◇ Actualización elitista de la feromona: La mejor solución encontrada hasta ahora actualizara la feromona para reforzar una intensificación en la búsqueda. Un valor mínimo (respectivamente un máximo) para la feromona es definido tal que para cada decisión haya una probabilidad mínima (resp. máxima) de ser elegido.

Adicionalmente a los componentes clásicos y comunes de búsqueda en metaheurísticas (por ejemplo, representaciones), el asunto principal en diseñar un ACO es la determinación de:

- Información de feromona: El modelo de feromona representa el componente central de los algoritmos ACOs. Este consiste en definir un vector de parámetros modelos τ llamados parámetros de camino de feromona. Los valores de feromona $\tau_i \in \tau$ deben reflejar la información relevante en la construcción de la solución para un problema dado. Ellos están usualmente asociado con los componentes de una solución.
- Construcción de solución: En la construcción de la solución, la pregunta principal esta concernida con la definición de la heurística local para orientar la búsqueda para la feromona. La metáfora ACO es fácilmente adaptable para resolver problemas donde algoritmos golosos relativamente eficientes existen.
- Actualización de la feromona: Principalmente la estrategia de aprendizaje de refuerzo para la información de las feromonas tiene que ser definida.

Ejemplo: ACO para TSP

Consideremos ahora la aplicación del algoritmo ACO al proble TSP. El sistema de Hormigas (AS) fue la primera aplicación de ACO en el dominio de la optimización combinatoria. Sea $G = (V, E)$ el grafo de entrada. Diseñar un algoritmo ACO para el TSP necesita de una buena definición del rastro de feromona y el proceso de construcción de la solución (Algoritmo 9).

Algorithm 9 Algoritmo ACO para TSP.

Inicializar la información de la feromona;
repeat
  for cada hormiga **do**
    Construcción de la solución usando el rastro de las feromonas:
    $S = \{1, 2, ..., n\}$ /* Conjunto de potenciales ciudades a escoger */
    Selección aleatoria de la ciudad inicial i;
    repeat
      Seleccionar una nueva ciudad j con probabilidad $p_{ij} = \frac{\tau_{ij}^{\alpha}\ \eta_{ik}^{\beta}}{\sum_{k \in S} \tau_{ij}^{\alpha}\ \eta_{ik}^{\beta}}$
      $S = S - \{j\};\ i = j$
    until $S = \emptyset$
  end for
  /* Actualizar los caminos de feromona*/
  for $i, j \in [1, n]$ **do**
    $\tau_{ij} = (1 - \rho)\tau_{ij}$; /*Evaporación*/
  end for
  for $i \in [1, n]$ **do**
    $\tau_{i\pi(i)} = \tau_{i\pi(i)} + \Delta$; /* π: Mejor solución encontrada */
  end for
until Criterio de detención
Output: Mejor solución encontrada o conjunto de soluciones

Cuadro 6: Resumen de Metaheurísticas.

Metaheurística	Clasificación	Tipo de Búsqueda	Búsqueda	Tipo de Memoria	S o P-Metaheurística
Tabú Search	Iterativa	No Prob	Estrategía de búsquedas globales	Corto y largo plazo	S-Metaheurística
Simulated Annealing	Iterativa	Prob	Estrategías de búsquedas globales	No tiene	S-Metaheurística
Algoritmo Genético	P-Metaheurística	Prob	Estrategías de búsquedas globales	No tiene	P-Metaheurística
Scatter Search	P-Metaheurística (híbrido entre TS y AE)	Prob	Inserto un procedimiento de búsqueda local	No tiene	P-Metaheurística
Colonia de Hormigas	Const. y con pobl.	Prob	estrategias de búsquedas globales	No tiene	P-Metaheurística

Metaheurística	Aleatoria?	Intensificación o diversificación	Con/sin Vecindario	Parámetros
Tabú Search	No	Ambas (Concepto propio)	Con vecindario	Criterio de intensificación, método de intensificación, criterio de diversificación, método de diversificación, criterio de detención, condición de aspiración, seteo de memoria de corto y largo plazo
Simulated Annealing	Si	Ambas(puede ser, es necesario probar)	Con vecindario	Programa de enfriamiento, T^0 inicial, criterio de detención,solución inicial, condición de equilibrio
Algoritmo Genético	si	Ambas (forzada, hay que mostrarlo)	Sin vecindario	Tamaño de población P, probabilidad de cruzamiento p_c, probabilidad de mutación p_m, criterio de termino (#iteraciones/tiempo max.)
Scatter Search	si		Con vecindario	Tamaño de población P, Criterio de detención, método de mejoramiento de la población, de generación de subconjuntos, de combinación de soluciones
Colonia de Hormigas	si		Sin vecindario	Cantidad de hormigas (tamaño de población), tasa de reducción de feromona ρ, método de actualización de feromona y de desvanecimiento, criterio de detención

Referencias

[Aarts et al., 2003] Aarts, E., Aarts, E. H., and Lenstra, J. K. (2003). *Local search in combinatorial optimization*. Princeton University Press.

[Althöfer and Koschnick, 1991] Althöfer, I. and Koschnick, K.-U. (1991). On the convergence of "threshold accepting". *Applied Mathematics and Optimization*, 24(1):183–195.

[Baum, 1986] Baum, E. (1986). Iterated descent: A better algorithm for local search in combinatorial optimization problems. *Manuscript*.

[Černỳ, 1985] Černỳ, V. (1985). Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm. *Journal of optimization theory and applications*, 45(1):41–51.

[Charles, 1859] Charles, D. (1859). On the origin of species by means of natural selection. *Murray, London*.

[Dorigo et al., 1996] Dorigo, M., Maniezzo, V., and Colorni, A. (1996). Ant system: optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 26(1):29–41.

[Fogel, 1962] Fogel, L. J. (1962). Toward inductive inference automata. In *Communications of the ACM*, volume 5, pages 319–319. ASSOC COMPUTING MACHINERY 1515 BROADWAY, NEW YORK, NY 10036.

[Fogel et al., 1966] Fogel, L. J., Owens, A. J., and Walsh, M. J. (1966). Artificial intelligence through simulated evolution.

[Glover, 1977] Glover, F. (1977). Heuristics for integer programming using surrogate constraints. *Decision sciences*, 8(1):156–166.

[Glover, 1989] Glover, F. (1989). Tabu search—part i. *ORSA Journal on computing*, 1(3):190–206.

[Hansen, 1986] Hansen, P. (1986). The steepest ascent mildest descent heuristic for combinatorial programming. In *Congress on numerical methods in combinatorial optimization, Capri, Italy*, pages 70–145.

[Holland, 1962] Holland, J. H. (1962). Outline for a logical theory of adaptive systems. *Journal of the ACM (JACM)*, 9(3):297–314.

[Holland, 1992] Holland, J. H. (1992). *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. MIT press.

[Johnson, 1990] Johnson, D. S. (1990). Local optimization and the traveling salesman problem. In *International colloquium on automata, languages, and programming*, pages 446–461. Springer.

[Kirkpatrick et al., 1983] Kirkpatrick, S., Gelatt, C. D., and Vecchi, M. P. (1983). Optimization by simulated annealing. *science*, 220(4598):671–680.

[Koza, 1994] Koza, J. R. (1994). Genetic programming as a means for programming computers by natural selection. *Statistics and computing*, 4(2):87–112.

[Martin et al., 1991] Martin, O., Otto, S. W., and Felten, E. W. (1991). Large-step markov chains for the traveling salesman problem.

[Metropolis et al., 1953] Metropolis, N., Rosenbluth, A. W., Rosenbluth, M. N., Teller, A. H., and Teller, E. (1953). Equation of state calculations by fast computing machines. *The journal of chemical physics*, 21(6):1087–1092.

[Rechenberg,] Rechenberg, E. Optimierung technischer systeme nach prinzipien der biologischen evolution. 1973. *Frommann-Holzboog Verlag, Stuttgart*.

[Rechenberg, 1965] Rechenberg, I. (1965). Cybernetic solution path of an experimental problem. *Royal Aircraft Establishment Library Translation 1122*.

[Schoen, 1991] Schoen, F. (1991). Stochastic techniques for global optimization: A survey of recent advances. *Journal of Global Optimization*, 1(3):207–228.

[Schott, 1995] Schott, J. R. (1995). Fault tolerant design using single and multicriteria genetic algorithm optimization. Technical report, AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH.

[Stützle, 1999] Stützle, T. (1999). Local search algorithms for combinatorial problems-analysis, algorithms and new applications. *DISKI-Dissertationen zur Künstlichen Intelligenz, Infix, Sankt Augustin, Germany*.

[Talbi, 2009] Talbi, E.-G. (2009). *Metaheuristics: from design to implementation*, volume 74. John Wiley & Sons.

[Talbi et al., 1998] Talbi, E.-G., Hafidi, Z., and Geib, J.-M. (1998). A parallel adaptive tabu search approach. *Parallel computing*, 24(14):2003–2019.