PONTIFICIA UNIVERSIDAD CATÓLICA DE VALPARAÍSO
FACULTAD DE INGENIERÍA
ESCUELA DE INGENIERÍA INFORMÁTICA

PROBLEMA DEL VENDEDOR VIAJERO PROBABILÍSTICO A TRAVÉS DE LAS METAHEURÍSTICAS PSO Y SA

JUAN PABLO RIQUELME REYES

INFORME FINAL DEL PROYECTO PARA OPTAR AL TÍTULO PROFESIONAL DE INGENIERO CIVIL EN INFORMÁTICA.

Marzo de 2011.

Pontificia Universidad Católica de Valparaíso
Facultad de Ingeniería
Escuela de Ingeniería Informática

PROBLEMA DEL VENDEDOR VIAJERO PROBABILÍSTICO A TRAVÉS DE LAS METAHEURÍSTICAS PSO Y SA.

Alumno: **JUAN PABLO RIQUELME REYES**

Profesor Guía: **Guillermo Cabrera Guerrero.**

Profesor Co-referente: **Silvana Roncagliolo De La Horra.**

Carrera: **Ingeniería Civil en Informática.**

Marzo de 2011.

Dedicatoria

“Dedicado a mi madre por su amor,
esfuerzo, dedicación y apoyo incondicional.”

Resumen

Resumen

El Problema del Vendedor Viajero Probabilístico (PTSP) es una variación del bien conocido Problema del Vendedor Viajero (TSP). Este problema surge, ya que la información no se encuentra disponible oportunamente para crear un calendario del Tour a seguir con las visitas a realizar y/o el costo de volver a rehacer el recorrido es muy elevado.

En esta investigación es propuesto un PTSP, donde su tour es estimado usando los algoritmos metaheurísticos Optimización de Enjambres de partículas (PSO) y Simulated Annealing (SA). La selección de PSO como técnica se justifica por ofrecer un algoritmo con estructura simple, con alta probabilidad de exploración del espacio de soluciones, y mayor velocidad de convergencia frente a otros algoritmos evolutivos. Se ha usado SA en este trabajo para incrementar la diversidad de las partículas, y de esta manera mitigar la fácil captura de mínimos locales por parte del PSO. Los resultados obtenidos mediante esta técnica híbrida se aproximan a los resultados obtenidos por su Benchmarck, lo que la hace un buen referente para la utilización.

Palabras Claves: Problema del Vendedor Viajero Probabilístico; Optimización de Enjambres de partículas; Simulated Annealing.

Abstract

The Probabilistic Traveling Salesman Problem (PTSP) is a variation of the well known Traveling Salesman Problem (TSP). This problem it's produced because the information it's not available in time to create a schedule for the tour, with the visits to make and/or the cost to redo the tour it's very high.

In this research a PTSP it's proposed, where its tour is estimated by using particle swarm optimization (PSO) and Simulated Annealing (SA) meta-heuristic algorithms. The selection of PSO as technique is justified because offers a simple structure algorithm, with high probability of exploration of solutions space, and faster convergence compared with others evolutive algorithms. SA was used in this jobs to increase the diversity of the particles, and thus mitigate the easy capture of local minima by PSO. Results obtained by this hybrid method are aproximed to results obtained by benchmarck, which make it a good reference for the use.

Keywords: Probabilistic Traveling Salesman Problem, particle swarm optimization, Simulated Annealing.

Índice de Contenido.

Glosario.

Circuito VLSI	:	**V**ery **L**arge **S**cale **I**ntegration. Integración en escala muy grande de sistemas de circuitos basados en transistores en circuitos integrados.
Ciclo de Hamilton	:	Sucesión de aristas adyacentes, que visita todos los vértices del grafo una sola vez.
Constricción	:	Reducción a límites menores.
Estocástico	:	Sistema que funciona, sobre todo, por el azar.
Factor de Boltzman	:	Factor de ponderación que determina la probabilidad relativa de un estado y en un sistema con múltiples estados en equilibrio termodinámico a temperatura *T*.
Metaheurística	:	Método heurístico para resolver un tipo de problema computacional general, usando los parámetros dados por el usuario sobre unos procedimientos genéricos y abstractos de una manera que se espera eficiente.
NP Completo	:	Tiempo polinónico no determinista completo.
PSO	:	Particle Swarm Optimization.
SA	:	Simulated Annealing.

Índice de Figuras.

Índice de Tablas.

CAPÍTULO 1

INTRODUCCIÓN

1.1 Introducción

Una gran cantidad de problemas importantes de optimización no pueden ser resueltos usando métodos exactos, es decir, no es posible encontrar su solución óptima con esfuerzos computacionales aceptables aunque se pueda contar con computadores de alta velocidad operando en paralelo. Un gran problema en optimización es el fenómeno llamado explosión combinatorial, en el cual, cuando crece el número de variables de decisión del problema, el número de decisiones factibles y el esfuerzo computacional crecen en forma exponencial.

Existe un creciente interés de la comunidad en la investigación de operaciones (IO), específicamente en el tratamiento de problemas de optimización, que incluyen en su formulación matemática incertidumbre, estocacidad, e información dinámica.

Resolución de problemas en condiciones de incertidumbre tiene un gran impacto en los contextos del mundo real, la optimización de los problemas que surgen en la práctica son cada vez más complejos y dinámicos y, también gracias a la rápida evolución de las telecomunicaciones que hace que no sólo la percepción, sino también los cambios del mundo sean más rápido, estocástico y difícil de predecir. Por lo tanto, el estudio de optimización de algoritmos para los problemas de optimización combinatorial estocástico (SCOPs) es un aspecto de la investigación de operaciones que está tomando una importancia creciente.

En los últimos años, los algoritmos metaheurísticos han surgido como alternativas a los clásicos enfoques basados en programación matemática y dinámica para la solución de SCOPs. De hecho, debido a la alta complejidad y la dificultad de problemas de optimización bajo incertidumbre, a menudo los enfoques clásicos (que garantizan a encontrar la solución óptima) son viables únicamente para los casos de tamaño pequeño, y podría requerir una gran cantidad de esfuerzo computacional. En cambio, los enfoques basados en metaheurísticas son capaces de encontrar buenas soluciones, a problemas de tamaño realista, por lo general, en un corto tiempo de cálculo.

El Problema del Vendedor Viajero Probabilístico (PTSP) es uno de los temas relevantes para las empresas debido a que tienen que enfrentarse a la gestión y coordinación de manera eficaz y efectiva un conjunto de rutas de entrega de uno o varios almacenes centrales a los diversos puntos de demandas que posee la Cadena de Abastecimiento. Particularmente la Gestión de la Cadena de Abastecimientos se puede definir como: “El conjunto de enfoques utilizados para integrar de manera eficiente a proveedores, productores, centros de almacenamiento y distribución, y vendedores minoristas, de modo

que los bienes sean producidos y distribuidos en las cantidades apropiadas, en los lugares correctos y en los instantes adecuados, minimizando los costos totales del sistema y satisfaciendo los niveles de servicio requeridos" (Lamas, 2007).

El PTSP es una variación del bien conocido Problema del Vendedor Viajero (TSP), que es un problema ampliamente estudiado en optimización combinatorial. En TSP estándar, el objetivo es encontrar la duración mínima del ciclo de Hamilton a través de un conjunto de n clientes, habiendo contado las distancias entre todos los pares de los clientes. En el PTSP, cada cliente i, además, tiene una probabilidad p_i de realizar una visita que es exigida. En lugar de simplemente reducir al mínimo la suma de las distancias de recorrido, la meta para PTSP es encontrar un recorrido, a priori, con una duración mínima prevista. (Bertsimas, 1988; Jaillet, 1985)

Debido al hecho de que el elemento de incertidumbre no sólo existe, sino que también afecta de manera significativa el rendimiento del sistema en el mundo real las aplicaciones de transporte y la logística, los resultados del PTSP pueden proporcionar información sobre la investigación en otros problemas de optimización combinatoria probabilística. Por otra parte, el PTSP también se puede utilizar para muchos modelos del mundo real en aplicaciones de logística y la planificación del transporte, tales como la entrega y/o recogida diaria de servicios con demanda estocástica, la secuencia de trabajo que impliquen cambio de coste, diseño de secuencias de recuperación en un almacén o en una terminal de operaciones de carga, servicios de reparto de comidas a la tercera edad, problema de ruteo de vehículos estocásticos de demanda, y servicio de entrega a domicilio en virtud de comercio electrónico (Bertsimas et al., 1995; Jaillet, 1988; Tang-Hooks y Miller, 2004).

PTSP representa un ejemplo de un modelo de planificación estratégica en la que Factores estocásticos se consideran explícitamente. Estos tipos de problemas que son de importancia central en logística y la planificación del transporte, donde las aplicaciones de optimización de nuevo puede ser inviable, ya sea por razones operativas o de cálculo. En el caso de PTSP, la optimización puede ser imposible o inoportuna, por ejemplo, debido a limitaciones de tiempo, porque la comunicación de las actualizaciones del tour no es posible.

Se propone en este trabajo de título un PTSP, donde su tour es estimado usando el algoritmo metaheurístico Optimización de Enjambres de partículas (PSO) y Simulated Annealing (SA). PSO propuesta por Kennedy y Eberhart en [12], es considerado uno de los métodos estocásticos basado en población más robusta de la computación evolutiva dentro de la categoría de inteligencia de enjambres. Este paradigma está inspirado en el comportamiento de grupos colectivos tales como enjambre de avispas y bandas de pájaros tratando de imitar la naturaleza social, organizativa y evolutiva, para poder resolver problemas complejos de optimización, diseño, predicción, control planificación, minería de datos, negocios y finanzas [7]. SA fue presentada por primera vez por Metrópolis y aplicado con éxito a los problemas de optimización de Kirkpatrick [8]; inspirada en la idea de las industrias en calentar a alta temperatura los materiales para luego hacer descender, así se consiguen materiales más resistentes o más cristalinos. La selección de PSO como técnica se justifica por ofrecer un algoritmo con estructura simple, con alta probabilidad de

exploración del espacio de soluciones, y mayor velocidad de convergencia frente a otros algoritmos evolutivos como ACO y, la selección de SA; en este trabajo es usado para incrementar la diversidad de las partículas y eliminar el estancamiento que produce el PSO en los mínimos locales.

1.2 Objetivos

1.2.1 Objetivo General

Resolver Problema del Vendedor Viajero Probabilístico (PTSP) a través de la hibridación metaheurística de enjambre de partículas (PSO) y Simulated Annealing (SA).

1.2.2 Objetivos Específicos

- Tener una visión acabada del estado del arte del PTSP y sus aplicaciones para que pueda ser representado el problema, de manera que pueda ser resuelto por alguna técnica estudiada, en este caso la metaheurística PSO y SA.

- Comprender el funcionamiento de la metaheurística de Enjambres de Partículas (PSO) y Simulated Annealing (SA).

- Crear un modelo de implementación para PSO.

- Evaluar y contrarrestar el rendimiento de la metaheurística propuesta con metaheurísticas vista en trabajos anteriores.

1.3 Organización del texto

Este documento está estructurado de la siguiente manera: El capítulo 2 se destina a la definición del TSP. El capítulo 3 se da una mirada profunda a la función de evaluación del PTSP; es decir la función utilizada para el tour, discutiendo las posibilidades de su aproximación. El capítulo 4 da a conocer el problema de optimización combinatorial estocástico el cual muestra un enfoque de modelado de la incertidumbre y la descripción formal de éste. El capítulo 5 da la definición de enjambre de partículas, los fundamentos del movimiento de partículas, el comportamiento social y el operador de velocidad y posición. El capitulo 6 da una definición de Simulated Annealing, sus fundamentos, su analogía física, y la descripción del algoritmo El capítulo 7 propone el modelo y diseño de la solución en base a la inicialización de los individuos, modo de selección de los vecinos, criterio de detención y otros parámetros que son necesarios para su desarrollo. El capítulo 8 hace una comparación de los algoritmos PSO vs. su hibrido PSO con SA para tomar como referencia en la resolución del problema. El capítulo 9, muestra resultados del TSP con Enjambre de Partículas y Simulated Annealing, resultados comparados con los Benchmarck de la literatura. El capítulo 10, muestra una implementación del PTSP con enjambre de partículas, mostrando los resultados obtenidos por éste. El documento concluye con el capítulo 11, en la cual da una conclusión de la posibilidad de resolución del PTSP con las metaheurísticas propuestas.

CAPÍTULO 2

PROBLEMA DEL VENDEDOR VIAJERO (TSP)

2.1 Estructura del Problema del Vendedor Viajero (TSP)

El Problema del Vendedor Viajero es uno de los problemas clásicos de optimización combinatorial, estudiado en la investigación de operaciones y la informática teórica, y consiste en encontrar un recorrido que, partiendo de la ciudad de origen, debe visitar exactamente una vez cada ciudad dentro de un conjunto finito de ellas (previamente especificadas) y retornar al punto de partida. Un recorrido de estas características es llamado tour. El problema consiste en encontrar el tour para el cual la distancia total recorrida sea mínima, asumiendo que se conocen las distancias entre las ciudades como se presenta en la Figura 1.

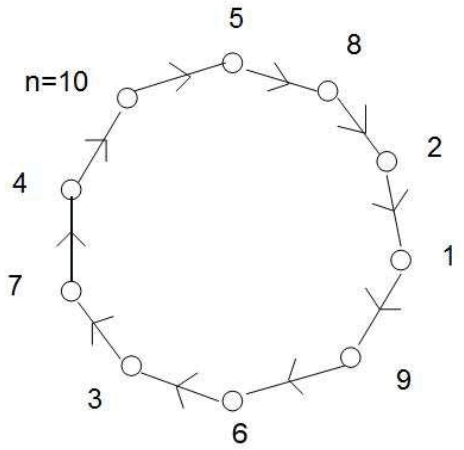


Figura 2.1. Estructura de costos del Problema del Vendedor Viajero.

Se pueden plantear dos tipos de TSP: simétrico y asimétrico. En el caso simétrico, la distancia de la ciudad i a la ciudad j es la misma que la distancia de la cuidad j a la ciudad i; es decir que los arcos (i,j) y (j,i) siempre existen o no existen simultáneamente en la cual su longitud es la misma. En el caso asimétrico las distancias pueden ser distintas dependiendo de la ciudad de partida por lo que no se obtendrá la misma longitud del tour al recorrerla en forma inversa como es en el caso del simétrico.

Figura 2.2. Ejemplo de TSP simétrico. La distancia de i a j es la misma que la distancia de j a i.

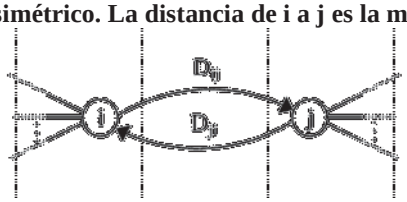


Figura 2.3. Ejemplo de TSP asimétrico. La distancia de i a j puede no ser la misma que la distancia de j a i.

El problema del Vendedor Viajero está clasificado como NP-hard ó NP-Completo (nodeterministic polynomial problem hard), es decir la dificultad aumenta de forma exponencial con el tamaño del problema a resolver [20].

Al ser TSP un problema NP-Completo, una de las dificultades es que no existen algoritmos que garanticen encontrar una solución óptima para cualquier instancia del problema en un tiempo que crezca en forma polinomial con el número de nodos, lo que dificulta su interacción con problemas de localización tradicionales.

Además del reto computacional que representa este tipo de problemas, también es de los más interesantes de la teoría de grafos y de la investigación operativa por su aplicación práctica en la realidad. Su aplicación es visible y de gran importancia para la resolución de problemas reales en la Dirección de Operaciones y Logística. Por ejemplo: problemas de picking, de rutas, sistemas de navegación GPS, planificación de movimientos de robots, vehículos autoguiados (AGV), etc.

Formalmente se define TSP de tamaño n de la siguiente manera: Sea $G=(V,A)$ un grafo dirigido con un conjunto V de vértices y un conjunto A de arcos. Se tiene que $V=N$, donde $N=\{0,1,...,n-1\}$ es el conjunto de ciudades y dentro de ellas se considera a 0 como el origen. Para el conjunto de arcos, se tiene que A es el conjunto de arcos que conecta a todas las ciudades entre sí. Para cada arco $(i,j)\in A$ se conoce una distancia d_{ij}.

Además de los parámetros conocidos, también se tiene la variable:
x_{ij} es 1 si es utilizado el arco (i,j), o 0 en otro caso.

Siendo el problema el determinar cuál de los arcos $(i,j)\in A$ será utilizado en la ruta.

El modelo matemático propuesto por Ronald L. Rardin es:

Minimizar.

$$C_{TSP}(n,\tau) = \sum_{i} \sum_{j>i} d_{ij} x_{ij} \quad \forall \text{ i } \in \text{ N} \tag{2.1}$$

Sujeto a:

$$\sum_{j<i} x_{ij} + \sum_{j>i} x_{ij} = 2 \quad \forall \text{ i } \in \text{ N} \tag{2.2}$$

$$\sum_{i \in S} \sum_{j \notin S,\, j>i} \text{x}_{\text{ij}} + \sum_{i \notin S} \sum_{j \in S} \text{x}_{\text{ij}} \geq 2 \quad \forall \text{ i } \in N; j > i; S \subset N, |S| \geq 3 \tag{2.3}$$

$$x_{ij} \in \{0,1\} \quad \forall \text{ i,j } \in \text{ N} \tag{2.4}$$

La ecuación (2.1) es la función objetivo a minimizar, donde queda expresada el costo de cada arco.

La ecuación (2.2) asegura que, desde todas las ciudades se continúa hacia alguna otra, y que a todas las ciudades se llega desde algún otro nodo.

La ecuación (2.3), da las restricciones que evita que se produzcan subciclos en la solución.

La ecuación (2.4), da la restricción que resguarda la integridad del modelo.

Analizando las restricciones, se puede observar que sólo debe haber un arco de llegada a una ciudad o nodo, y que igualmente tan sólo debe haber un arco de salida, con esto se garantiza que cada ciudad sea visitada sólo una única vez.

Si bien es cierto que la estructura TSP representa de mejor forma el comportamiento real que tienen los vehículos al momento de distribuir al interior de un cluster, aún hay aspectos que no están representados en el TSP, como la probabilidad de que algunos clientes deban o no formar parte del tour. En otras palabras, dado un conjunto de clientes conocidos, no necesariamente serán visitados todos durante la misma jornada o periodo de distribución, puesto que ellos podrían no solicitar productos en todos los períodos.

2.2 Algoritmos de resolución del TSP

Las formas tradicionales de tratar este problema son: encontrar una solución exacta, cuando las dimensiones del problema son relativamente pequeñas, utilizar algoritmos heurísticos, lo cual proporciona buenas soluciones, aunque no pudiendo garantizar siempre que sean las óptimas; o bien, dividir el problema en subproblemas y reducir así la dificultad de éstos.

Un desarrollo más elaborado es la aplicación de algunas metas heurísticas como la de Colonia de Hormigas (ACO), Simulated Annealing (SA), Algoritmo genético (GA), etc.

2.2.1 Algoritmos exactos

TSP se puede fácilmente visualizar que es un problema asociado a la determinación de un Ciclo Hamiltoniano (HC) en un grafo completo (un ciclo que recorra todos los vértices, pasando una sola vez por cada uno de ellos).

Si se quisiera solucionar el problema de HC de manera óptima, en el peor de los casos se debería revisar el total de las combinaciones posibles de sus vértices. Si se posee n vértices, la cantidad total de combinaciones distintas a revisar sería de $\frac{n!}{2}$, por lo que si se dispusiera de un computador capaz de realizar un millón de pasos por segundo, el tiempo que demoraría en revisar el total de las soluciones posibles para distintos valores de n se puede apreciar en la tabla 2.1.

TABLA 2.1. AUMENTO DEL TIEMPO SEGÚN AUMENTO DE N.

N	Combinaciones	Tiempo Requerido	
1	1	0,000001	Segundos
2	1	0,000001	Segundos
3	3	0,000003	Segundos
4	12	0,000012	Segundos
5	60	0,00006	Segundos
6	360	0,00036	Segundos
7	2520	0,00252	Segundos
8	20160	0,02016	Segundos
9	181440	0,18144	Segundos
10	1814400	1,8144	Segundos
11	19958400	19,9584	Segundos
12	239500800	3,99168	Minutos
13	3113510400	51,89184	Minutos
14	43589145600	12,108096	Horas
15	6,53837E+11	7,56756	Días
16	1,04614E+13	121,08096	Días
17	1,77844E+14	5,639387178	Años
18	3,20119E+15	101,5089692	Años

2.2.2 Heurísticas

Se han desarrollado varios algoritmos heurísticos, que proporcionan rápidamente buenas soluciones. Se pueden encontrar soluciones para problemas extremadamente largos en un tiempo razonable, con una desviación del óptimo de un 2 ó 3%. [Applegate, 2006].

CAPÍTULO 3

PROBLEMA DEL VENDEDOR VIAJERO PROBABILÍSTICO (PTSP)

3.1 Definición y Evaluación del PTSP

Para muchas empresas de entrega sólo un subconjunto de los clientes requiere una recolección o entrega cada día, lo cual resulta costoso hacer un TSP para cada día debido a que:

- La información puede no estar disponible con suficiente antelación para crear calendarios de cada día óptimo para aquellos clientes que requieren una visita (falta de información previa) ó;
- El costo de adquirir suficiente potencia de cálculo para encontrar esas soluciones puede ser demasiado elevado. Por ésta razón, es usual diseñar un recorrido que contenga todos los clientes (llamado tour a priori), y cada día sigue el orden del tour, a priori, visitando solo a los clientes que requieren de una visita ese día (ver figura 3.1).

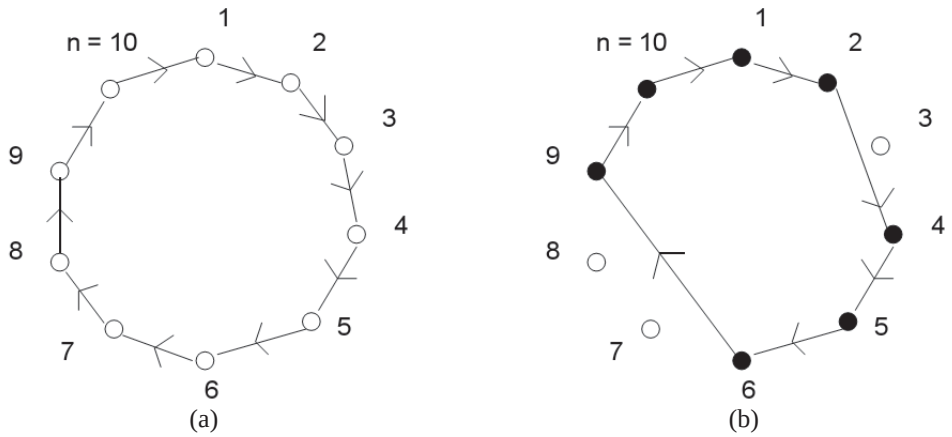


Figura 3.1. Ejemplo de un viaje a priori (a la izquierda [a]), y una posible visita a posteriori (a la derecha [b]) obtenidos a partir del tour, a priori, visitando sólo los clientes que requieren una visita al azar en un determinado momento, y manteniendo el mismo orden de visita como en el recorrido a priori. En este ejemplo, al azar, donde sólo los clientes número 1, 2, 4, 5, 6, 9, 10 requieren una visita.

El problema de encontrar, a priori, un tour de costo mínimo, dado un conjunto de clientes donde cada uno de ellos tiene una determinada probabilidad de requerir una visita, se define el PTSP. [2]

La principal diferencia entre PTSP y TSP es que en el PTSP la probabilidad de cada nodo que se visita es entre 0,0 y 1,0; mientras que en TSP la probabilidad de cada nodo que se visita es de 1 ó 0. [17]

Debido al hecho de que el elemento de incertidumbre no sólo existe, sino que también afecta de manera significativa el rendimiento del sistema en el mundo real, las aplicaciones de transporte y la logística, los resultados del PTSP pueden proporcionar información sobre la investigación en otros problemas de optimización combinatorial probabilístico.

El PTSP se define, según [16], como un grafo dirigido $G = (V, E)$, donde $V := (0, v_1, v_2, ..., v_n)$ es el conjunto de nodos o vértices, $E \in V \times V$ es el conjunto de aristas dirigidas. El nodo 0 representa el depósito con la presencia de probabilidad de 1,0. Cada nodo que no sea el nodo de depósito v_i $(i = 1, 2, ..., n)$ se asocia con una probabilidad de presencia de p_i, que representa la posibilidad de que el nodo v_i estará presente en una realización[1] dada. Entregado un grafo dirigido G, el objetivo del PTSP es encontrar un recorrido, a priori, con un mínimo de duración prevista en G.

3.2 Estructura del Problema del Vendedor Viajero Probabilístico

La solución del PTSP se basa principalmente en el cálculo de la duración prevista de una visita a priori. El cálculo de la duración prevista de una visita a priori τ del PTSP, denominado E[τ], depende de la ubicación relativa de los nodos del giro y la probabilidad de presencia de cada uno de los nodos en una determinada instancia. Al examinar todas las realizaciones expresamente sobre la base de la presencia de cada nodo, la duración prevista del viaje τ se puede calcular.

Si se tiene un total de n clientes, el número total de realizaciones posibles del problema es 2^n. La probabilidad de realización de r_i, $p(r_i)$ (con $p(r_i) \in [0,1]$), se puede calcular sobre la base de la probabilidad de presencia de cada nodo. $L[r_j(\tau)]$ describe la duración del recorrido τ para la realización r_i en el supuesto de que los nodos no son simplemente saltados del giro r_i. La duración esperada del giro puede ser descrito formalmente como:

$$E(\tau) = \sum_{i=1}^{2^n} p(r_i) \cdot l[r_i(\tau)]. \qquad (3.1)$$

[1] Una realización corresponde a un subconjunto del total de clientes que deben ser visitados de manera sincrónica en un mismo periodo (jornada) de distribución.

El cálculo de la duración prevista sobre la base de la ecuación (3.1) es ineficiente, ya que la complejidad computacional aumenta exponencialmente con un aumento del número de nodos.

Más formalmente, se define $N = \{i / i = 1, 2, ..., n\}$ un conjunto de n clientes. Para cada par de clientes $i, j \in N, d_{ij}$ representa la distancia entre i y j. En este caso, se tomará como supuesto que las distancias (ó costes) son simétricas, es decir, $d_{ij} = d_{ji}$. Un tour a priori de $\tau = (\tau(1), \tau(2), ..., \tau(n))$ es una permutación sobre N, es decir, una visita a todos los clientes exactamente una vez. Dada la probabilidad independiente p_i que el cliente i requiere una visita, $q_i = 1 - p_i$ es la probabilidad de que i no requiera una visita. El caso general, donde los clientes probabilidades p_i pueden ser diferentes, se denomina PTSP heterogéneo, mientras que si las probabilidades son iguales entre sí ($p_i = p$ para cada cliente i), el problema se llama PTSP homogéneo. Se utilizará la siguiente convención para cualquier cliente índice i:

$$i := \begin{cases} i \bmod n, & si \;\; i \neq 0 \; y \; i \neq n \\ n, & en \; otro \; caso \end{cases} \tag{3.2}$$

Donde i mod n es el resto de la división de i por n. La razón para definir el anterior convenio es que se quiere utilizar como cliente de los índices de los números 1 a n (y no de 0 a n-1), teniendo que hacer cálculos con el operador 'módulo'. La duración prevista de un viaje, a priori, τ puede ser calculada en $O(n^2)$ con la siguiente expresión derivada por Jaillet [11]:

$$E[L(\tau)] = \sum_{i=1}^{n} \sum_{r=1}^{n-1} \left\{ d_{\tau(i)\tau(i+r)} p_{\tau(i)} p_{\tau(i+r)} \prod_{i+1}^{i+r-1} q_{\tau} \right\} \tag{3.3}$$

Utilizamos la siguiente notación para cualquier $i, j \in \{1, 2, ..., n\}$:

$$\prod_{i}^{j} q_{\tau} := \begin{cases} \prod_{t=i}^{j} q_{\tau}(t) & \text{si } 0 \leq \text{j-i<n-1} \\ \prod_{t=i}^{n} q_{\tau}(t) \prod_{u=1}^{j} q_{\tau}(u) & \text{si i-j>1} \\ 1 & \text{en otro caso.} \end{cases} \tag{3.4}$$

La expresión de la función objetivo (Ecuación (3.3)) tiene la siguiente explicación intuitiva: cada término en la suma representa la distancia entre el cliente i y el cliente ponderado

$(i+r)$ por la probabilidad de que los dos clientes requieren de una visita $\left(p_\lambda(i)p_\lambda(i+r)\right)$ mientras que los r-1 clientes no requieren de una visita $\left(\prod_{i+1}^{i+r-1} q_\lambda\right)$.

Para distribuciones de probabilidad homogénea, donde $p_i = p$ y $q_i = q$ para cada cliente i, la expresión de la función objetivo todavía requiere un tiempo de orden $O(n^2)$ de cálculo, pero es un poco más simple que la ecuación (3.3) siendo definida como:

$$E[L(\tau)] = \sum_{i=1}^{n} \sum_{r=1}^{n-1} p^2 q^{r-1} d_{\tau(i)\tau(i+r)} \tag{3.5}$$

Tenga en cuenta que el PTSP cae en la categoría de SIPs (Stochastic Integer Programs), ésta es parte de los problemas de optimización combinatorial estocástica el cual trata de resolver a través de metaheurísticas este tipo de problemas. [3]

CAPÍTULO 4

PROBLEMAS DE OPTIMIZACIÓN COMBINATORIAL ESTOCÁSTICA (SCOPs)

Hay un creciente interés de la comunidad científica en el tratamiento de problemas de optimización que incluyan en su formulación matemática incierta, estocástica e información dinámica. La resolución de problemas en situaciones de incertidumbre tiene un impacto muy alto en el mundo real, ya que los problemas de optimización que surgen en la práctica son cada vez más complejos y dinámicos, gracias al rápido desarrollo de las telecomunicaciones que hace que no sólo la percepción, sino también los cambios del mundo sean más rápidos, estocásticos y difíciles de pronosticar. Un grupo importante de problemas de optimización combinatoria en condiciones de incertidumbre corresponde a la clase de problemas de optimización combinatoria estocástica (SCOPs). El rasgo distintivo de SCOPs es que parte de la información sobre los datos del problema son desconocidos, y el conocimiento sobre su distribución de probabilidad es asumido.

Con el fin de darle una organización de los posibles enfoques de modelado de optimización bajo incertidumbre es que [3] propone una clasificación donde, la posición de este tipo de problemas, puede ser vista en perspectiva. Luego, algunas de las principales definiciones formales de SCOPs estático y dinámico de la literatura son presentadas.

4.1 Enfoques de modelado de la incertidumbre

Al definir el alcance de SCOPs se debe considerar la posibilidad de las muchas formas en que la incertidumbre puede ser formalizada. La incertidumbre es incluida en la formulación de problemas de optimización con el fin de ir más cerca de la condiciones del mundo real, pero también los modelos deben ser un poco simplificados, para que sea manejable analítica o numéricamente. El esfuerzo realizado para llegar a una buena relación entre la utilidad del modelo y la docilidad (ó maleabilidad) del problema ha producido una multitud de formalizaciones de incertidumbre. Esto es aún más evidente en el caso de las metaheurísticas, porque, debido a su sencillez, pueden ser fácilmente aplicadas a complejas fórmulas que se consideran difíciles para muchos enfoques de algoritmos clásicos.

TABLA 4.1. EVIDENCIA DE ALGUNAS DE LAS VENTAJAS EN LA SOLUCIÓN DE SCOPS, MOSTRADOS EN [3], A TRAVÉS DE METAHEURÍSTICAS EXACTAS EN LUGAR DE UTILIZAR LOS MÉTODOS CLÁSICOS. LAS METAHEURÍSTICAS SON CAPACES DE ENCONTRAR BUENAS SOLUCIONES Y, A VECES, A PROBLEMAS DE TAMAÑO REALISTAS, POR LO GENERAL, EN UN CORTO TIEMPO DE CÁLCULO.

Referencia (s)	**SCOP**	Metaheurística(**s**)	**Método exacto**	**Ventaja de la(s) metaheurística(s) sobre el método exacto**
Bianchi et al. (2005)	VRPSD	ACO, SA, TS, ILS, EC	Método entero L-shaped por Gendreau et al. Resuelto con hasta 70 clientes.	ACO, SA, TS, ILS, EC aborda los casos con un máximo de 200 clientes (a partir de la distancia óptima desconocido)
Branke y Guntsch (2003), Bianchi (2002)	PTSP	ACO, SA	Branch and Cut por Laporte resuelve instancias con hasta 50 clientes.	En Branke y Guntsch , ACO aborda instancias con hasta 200 clientes. En Bianchi, ACO aborda instancias de hasta 400 clientes.

Al considerar los modelos de problemas de optimización bajo incertidumbre, hay principalmente dos aspectos a definir: en primer lugar, el camino incierto de información oficial, y en segundo lugar, el dinamismo del modelo; es decir, el momento de la información revelada es incierto con respecto al momento en que las decisiones deben tomarse. El modelado de varios enfoques difiere en la forma en que el primero o el segundo definen los aspectos.

La información incierta puede ser formalizada de varias maneras. En el caso de un perfecto conocimiento acerca de los datos del problema corresponde a la esfera de la solución clásica de los Problemas (deterministas) de optimización combinatoria (DCOPs). Aquí, toda la información está disponible en la etapa de decisión, y es utilizada por los algoritmos de optimización para encontrar una posible solución óptima. La aplicación concreta de encontrar una solución conduciría exactamente al costo de la solución calculada por el algoritmo de optimización, por lo tanto también se considera como problema DCOPs estático, porque desde el punto de vista de la toma de decisiones no hay nada más que decir después de que se llevó a cabo la optimización. Un ejemplo típico de DCOP es el conocido Problema del Vendedor Viajero (TSP), donde, dado un conjunto de clientes y el conjunto de valores de distancia entre cada par de clientes, se debe encontrar el ciclo hamiltoniano (es decir, una visita una vez por cliente) de longitud mínima. A pesar de su simple formulación, el TSP es un problema NP-Completo, al igual que muchos DCOPs.

En SCOPs se puede distinguir un tiempo antes de la realización de las variables aleatorias, y un tiempo después de que las variables aleatorias son reveladas, ya que los sucesos aleatorios asociados suceden. Los SCOPs se caracterizan por el hecho de que las decisiones, o, equivalentemente, la identificación de una posible solución óptima es realizada antes de la realización de las variables aleatorias. Este marco se aplica cuando una determinada solución se puede aplicar sin modificaciones (o muy pequeñas) una vez que la realización efectiva de las variables aleatorias se conocen. La literatura a veces se ocupa de este tipo de problemas como optimización 'a priori'. Como ejemplo de esta clase de problemas, tenga en cuenta los TSP probabilistas (PTSP), que consiste en la búsqueda de una visita hamiltoniana a todos los clientes, de mínimo costo, dado que cada cliente tiene

una probabilidad conocida de exigir una visita. Una vez que la información de los clientes que realmente requieren una visita de un día determinado es conocida, los clientes que requieren de una visita son visitados en el orden "a priori" del viaje, y los que no requieren una visita, simplemente son saltados *(Bianchi, 2006)*.

Los SCOPs dinámico surgen cuando no es posible o no es conveniente el diseño de una solución que se puede usar en cualquier realización de las variables aleatorias. En este caso, las decisiones que necesiten un esfuerzo de optimización deben tomarse después de que también han ocurrido sucesos aleatorios. Esto se puede hacer en etapas, porque a menudo la información de incertidumbre no se revela toda a la vez. Como ejemplo de SCOP dinámico, consideramos, por ejemplo, un TSP donde conoce a nuevas posiciones de clientes los cuales aparecen con una cierta probabilidad, mientras que el vendedor ya ha comenzado a visitar los clientes "a priori". En este caso, "a priori", el Tour debe ser modificado de forma dinámica a fin de incluir a los nuevos clientes en el recorrido de visita.

TABLA 4.2. EXPLICACIÓN DEL USO DE ACRÓNIMOS PARA REFERIRSE A ALGUNOS SCOPS RELEVANTES.

Acrónimo	Nombre completo de SCOP
PTSP	Problema del Vendedor Viajero Probabilístico.
TSPTW	Problema del Vendedor Viajero con tiempo de ventanas estocásticas.
VRPSD	Problema de vehículos de enrutamiento con demanda estocástica y clientes.
SOPTC	Problema de orden secuencial con limitación de tiempo.

Los SCOPs son problemas de optimización combinatoria, es decir, problemas en que el espacio de decisión es finito pero posiblemente demasiado grande para ser enumerado, y/o problemas que tengan una estructura combinatoria, porque las soluciones están codificadas por permutaciones, vectores binarios u otros objetos combinatorios. Este abarca muchos contextos prácticos, tales como problemas de enrutamiento de vehículos, donde la estocasticidad se debe a la demanda variable de los clientes y la velocidad de los paquetes de información, la financiación, programación, ubicación de problemas y muchos otros contextos. Todos estos dominios pueden ser problema, y por lo general son, como también el modelo DCOPs. La ventaja de usar más SCOPs que DCOPs es que las soluciones pueden ser producidas de manera más sencilla y de mejor adaptación a las situaciones prácticas. Por supuesto, el uso de SCOPs en lugar de DCOPs tiene un precio: en primer lugar, la función objetivo suele ser mucho más exigente en SCOPs computacional que en DCOPs, en segundo lugar, para una aplicación práctica de SCOPs, existe la necesidad de evaluar las distribuciones de probabilidad de datos reales o subjetivos, una tarea que dista mucho de ser trivial.

4.2 Descripción formal de SCOPs

En este capítulo se tomará una definición general de SCOPs dada por Bianchi en [3] que es necesario para su comprensión.

Definición 1: Considere la posibilidad de un espacio de probabilidad (Ω, Σ, P), cuando Ω es el dominio de variables aleatorias ω(suele ser un subconjunto de R^k), Σ es una familia de "eventos", que es un subconjunto de Ω , y P es una probabilidad de distribución sobre Σ, con $P(\Omega) = 1$. Considere también un conjunto finito S de variable de decisión x. S es típicamente un subconjunto de R^n La variable aleatoria ω también podría depender de la variable de decisión x, que en caso de que se denote por ω_x. Dada la función de coste G y las funciones de limitación H_i, $i = 1, 2, ..., m$ limitado $(x, \omega) \in (S, \omega)$ sobre $\Re$.

$$\begin{cases} \text{"min"}_{x \in S} \quad G(x, \omega), \\ sujeto \quad a \quad H_i(x, \omega) \leq 0; \quad i = 1,2,...,m \end{cases} \tag{4.1}$$

Nótese que de acuerdo a [3], la definición anterior de SCOP no está bien definida, ya que el significado de "min", así como de las limitaciones no son claras en absoluto. De hecho, se hace imposible poder tomar una decisión en ω antes de saber el valor de x, y no se podría comprobar $H_i(x, \omega) \leq 0$ si todavía no se conoce. Además, dado que ω es una variable aleatoria, también $G(x, \omega)$ y $H_i(x, \omega)$ son variables aleatorias. Por estas razones, la definición de SCOPs debe ser refinada. Hay varias posibilidades de hacerlo, dando lugar a diferentes variantes de SCOP, tanto estáticas como dinámicas.

4.2.1 SCOPs estáticos

Definición 2: (Programación entera estocástica [Stochastic Integer Programa]-SIP) Dado un espacio de probabilidad (Ω, Σ, P), un conjunto finito S de soluciones de viable x, una función de coste real de un valor $G(x, \omega)$ de las dos variables $x \in S$ y $\omega \in \Omega$, y que denota por $\mathrm{E}_p\left(G(x, \omega)\right)$ el valor esperado de $G(x, \omega)$ sobre en Ω con arreglo a P, se denota por:

$$\min_{x \in S} \left\{ g(x) := \mathrm{E}_p\left(G(x, \omega)\right) \right\} \tag{4.2}$$

La definición anterior es quizás la más simple formulación de SCOP, y no se considera las limitaciones de azar.

En algunos casos, la función de coste G es determinista, es decir, G sólo depende de x y no de la variable aleatoria Ω, pero hace limitar la dependencia de la variable aleatoria ω. En tal situación sería imposible de aplicar $H_i(x, \omega) \leq 0$ para todos los $\omega \in \Omega$. Así, se puede relajar el concepto de satisfacción de restricciones, al permitir limitación de violaciones, y mediante la imposición de limitaciones que se cumplen al menos con algunas probabilidades dadas.

Definición 3: (Programa de cambio entero limitado [Change Constrained Interger Program] – CCIP) Dado un espacio de probabilidad (Ω, Σ, P), un conjunto finito S de soluciones viables x, una función de coste real de un valor $G(x)$, un conjunto de funciones limitado de valores reales $H_i(x, \omega)$, y un conjunto de probabilidades limitadas de violaciones α_i, con $0 \le \alpha_i \le 1$ e $i = 1, 2, ..., m$, encontramos

$$\begin{cases} \min\limits_{x \in S} \quad G(x), \\ \\ sujeto \quad a \quad \Pr ob\{H_i(x, \omega) \le 0\} \ge 1 - \alpha_i, \quad i = 1, 2, ..., m. \end{cases} \tag{4.3}$$

4.2.2 SCOPs dinámicos

Informalmente, un problema dinámico estocástico es un problema cuando las decisiones son tomadas en tiempos discretos $t = 1, ..., T$, el horizonte T puede ser finito o infinito. Las decisiones tomadas en el tiempo t pueden influenciar eventos aleatorios que suceden en el medio ambiente después del tiempo t. En los SCOPs dinámicos el concepto de la solución usada en SCOPs estáticos ya no es válido. Por ejemplo, en el TSP dinámico, un tour entre el conjunto de clientes conocidos al comienzo del día no puede ser recorrido, ya que en la práctica, debe ser modificado cuando las nuevas observaciones (nuevos clientes) son conocidas. Las decisiones se pueden hacer después del comienzo del proceso de toma de decisión para decidir qué política (o estrategia) adoptar, es decir, para especificar un conjunto de normas que digan qué medidas se tomarán al azar para cada posible acontecimiento futuro. Por ejemplo, en el TSP dinámico, una posible política consiste en el restablecimiento de la optimización de la parte de ruta entre los que aún no han visitado los clientes cada vez que aparece un nuevo cliente. Otra política, que es menos costosa, pero que eventualmente lleva a un tour más costoso, consiste en volver a optimizar en las fases, sólo después de que un cierto número de nuevos clientes ha aparecido. Se debe tener en cuenta que la solución de un SCOP dinámico tiene que hacer un doble esfuerzo: en primer lugar, decidir la política a adoptar, en segundo lugar, teniendo la política, resolver los sub-problemas de optimización que emergen dinámicamente. Ambas partes tienen una incidencia en el costo final de solución, pero a menudo la elección de la política se debe a factores que están fuera del control de la toma de decisiones. Por ejemplo, en el TSP dinámico no está obligado a optimizar cada vez que llega un nuevo cliente, en el caso de que los clientes quisieran saber con antelación la hora de llegada del vehículo.

4.2.3 Otro enfoque para metaheurística SCOPs

En los capítulo 5 y 6 se presentan la metaheurísticas de Optimización de Enjambres de Partículas (PSO) y Simulated Annealing (SA); las cuales de forma híbrida se pretende usar como método efectivo para la optimización combinatoria, en general, y en particular para SCOPs.

CAPÍTULO 5

OPTIMIZACIÓN POR ENJAMBRE DE PARTÍCULAS (PSO)

Los algoritmos metaheurísticos constituyen uno de los campos de investigación más activos en optimización. Su progreso está siendo rápido, surgiendo continuamente nuevas ideas en un intento de alcanzar una mayor eficiencia en el proceso de solución.

Actualmente los métodos de optimización heurísticos basados en poblaciones han despertado el interés de la comunidad científica debido a su capacidad para explorar espacios de soluciones multimodales y multidimensionales, de manera rápida y eficiente. La optimización por enjambre de partículas (PSO) pertenece a las técnicas estocásticas de cálculo evolutivo, considerada unas de las representantes de la rama de la inteligencia de enjambre (Swarm Intelligence), al igual que la técnica de optimización por colonia de hormigas (ACO), los cuales son los métodos más populares y utilizados en el área de la inteligencia computacional. Ambos tratan de imitar los comportamientos sociales de un colectivo a partir de la interacción de los individuos entre sí y con el entorno [7].

La optimización con enjambre de partículas, más conocida en la literatura científica como Particle Swarm Optimization (PSO), nace, al igual que otras técnicas estocásticas de cálculo evolutivo. Éste es uno de los métodos más populares y utilizados en el área de la inteligencia computacional, tratando de imitar el comportamiento social de los individuos a partir de la interacción de estos entre sí y el entorno.

Los orígenes del PSO como método estocástico de optimización global se remontan a los estudios iniciados por Kennedy y Eberhart en [12], quienes se fijan como objetivo inicial simular gráficamente el movimiento sincronizado e impredecible de grupos tales como los bancos de peces o las bandadas de aves, intrigados por la capacidad de estos grupos para separarse, reagruparse o encontrar alimento.

En un sistema PSO, la búsqueda se realiza utilizando una población de partículas que corresponden a los individuos, cada uno de los cuales representa una solución candidata al problema. Las partículas cambian su estado al volar a través del espacio de búsqueda hasta que se ha encontrado un estado relativamente estable. Esta metaheurística combina un modelo únicamente Social, el cual sugiere que los individuos ignoran su propia experiencia y ajustan sus conocimientos de acuerdo a las creencias exitosas de los individuos en la vecindad; y un modelo únicamente cognitivo el cual trata a los individuos como seres aislados.

En la terminología utilizada, Kennedy y Eberhart [12], introducen el término general partícula o agente para representar a los peces, pájaros, abejas, hormigas o cualquier otro tipo de individuos que exhiban un comportamiento social como grupo, en forma de una colección de agentes que interactúan entre sí. De acuerdo con los fundamentos teóricos del método, el movimiento de cada una de estas partículas hacia un objetivo común en dos dimensiones está condicionado por dos factores básicos, la memoria autobiográfica de la partícula o nostalgia y la influencia social de todo el enjambre. A nivel computacional, como método de optimización, esta filosofía puede extenderse a un espacio N-dimensional de acuerdo con el problema bajo análisis. La posición instantánea de cada una de las partículas de la población en el espacio N-dimensional representa una solución potencial, siendo N el número de incógnitas del problema original. Básicamente, el proceso evolutivo se reduce a mover cada partícula dentro del espacio de soluciones con una velocidad que variará de acuerdo a su velocidad actual, a la memoria de la partícula y a la información global que comparte el resto del enjambre, utilizando una función de fitness para cuantificar la calidad de cada partícula en función de la posición que ésta ocupe.

Más allá de la propia naturaleza del método, los esquemas existentes para su implementación son muy diversos. Dependiendo de cómo se actualicen las posiciones de las partículas surgen las versiones síncrona y asíncrona del algoritmo. Adicionalmente dependiendo de cómo se haga fluir la experiencia acumulada por el enjambre sobre el movimiento de cada una de las partículas que lo integran, se puede distinguir entre PSO local y global. La combinación de estas cuatro variantes resume los esquemas desarrollados e investigados comúnmente, pero en la literatura existen múltiples variantes a los esquemas convencionales, en la mayoría de los casos fruto de modificaciones introducidas por los propios autores para mejorar el rendimiento del algoritmo original en aplicaciones concretas.

Desde el punto de vista de su algoritmo, la ventaja principal es su rápida convergencia, en comparación con otros algoritmos de optimización global como los algoritmos genéticos (GA), colonia de hormigas (ACO), Simulated Annealing (SA), y otros algoritmos de optimización global [10]. Para aplicar PSO con éxito, uno de los problemas claves se encuentra en cómo representar la solución del problema en las partículas. Esta representación de la solución del problema afectará directamente la facilidad del uso del algoritmo y su funcionamiento [13].

5.1 Fundamentos del movimiento de Partículas

Para aplicar el movimiento de partículas a la vida artificial se deben respetar cinco principios básicos de lo que se entiende como inteligencia de grupo, más comúnmente recogido en la literatura como swarm intelligence [12], [19]. Estos principios reciben el nombre de ***proximidad, calidad, diversidad de respuesta, estabilidad y adaptabilidad***.

- **Principio de proximidad:** La población debe ser capaz de realizar cálculos sencillos de espacio y tiempo, lo cual se traduce en PSO a movimientos en N

dimensiones llevadas a cabo durante una serie de intervalos de tiempo que coinciden con movimientos de la población a una determinada velocidad.

- **Principio de calidad:** Promueve la capacidad de la población para responder a factores de calidad en el espacio de soluciones, lo que se consigue con la memoria de la partícula y con la historia o conocimiento social que comparten entre sí todos los congéneres.

- **Principio de diversidad de respuesta:** Promueve la diversidad de respuesta dentro de la población, y está garantizado por las diferentes tendencias marcadas por la memoria personal de cada partícula y por la historia de la mejor posición visitada por todo el conjunto.

  Los cuarto y quinto principios resaltan aspectos contrapuestos.

- **Principio de estabilidad:** La población sólo cambia su comportamiento como grupo cuando se actualiza la mejor posición históricamente visitada por alguno de los miembros que lo integran.

Por el contrario, de acuerdo con el

- **Principio de adaptabilidad:** La población debe ser a su vez adaptativa, es decir, debe ser capaz de modificar su comportamiento y movimiento cuando hay alguna señal que así lo recomienda desde el punto de vista de ahorro computacional o de mejora en la precisión. Esta premisa se consigue fácilmente, dado que la población en su conjunto cambia su rumbo cuando alguna de las partículas alcanza una solución global que mejora el resultado.

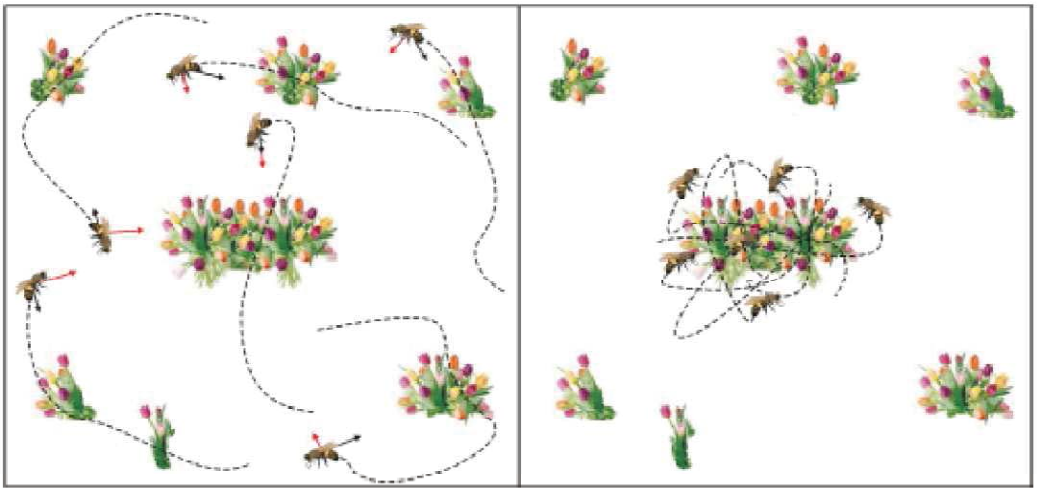
5.2 El Comportamiento social como método de Optimización Global

En PSO se usa el término multitud, enjambre y de forma genérica swarm, para referirse a cualquier conjunto de agentes o individuos que interactúan entre sí y con el medio que les rodea. Un ejemplo clásico de enjambre es representado por abejas en las inmediaciones de la colmena, haciéndose extensible a cualesquiera otros sistemas con una arquitectura y comportamiento social como grupo similar.

Para entender el funcionamiento del PSO como algoritmo de optimización supongamos, se usará a modo de ejemplo, el comportamiento que exhibe un enjambre de abejas en su movimiento sobre un campo cubierto con diferentes concentraciones de flores. Sin ningún conocimiento a priori del espacio de búsqueda, las abejas inician su movimiento desde posiciones aleatorias y con velocidades aleatorias. En su desplazamiento, el objetivo del enjambre se centra en encontrar el emplazamiento con la mayor densidad de flores. Cada abeja tiene memoria y puede recordar la posición visitada con mayor densidad de flores y también conoce, por mecanismos de comunicación con sus congéneres, la localización

donde otras abejas encontraron una densidad de flores significativa. Esta dupla de información es utilizada por la abeja para modificar continuamente su trayectoria, acelerando en ambas direcciones y volando hacia un punto espacial intermedio que dependerá de su posición actual y de cómo influyan sobre su decisión las así denominadas nostalgia o memoria y cooperación o conocimiento social. De esta forma, las abejas se encuentran permanentemente sobrevolando el campo en busca de posiciones con mayor densidad de flores, redirigiendo en parte la trayectoria del enjambre cada vez que se encuentran configuraciones de mayor calidad. Con el transcurso del tiempo, una vez ha sido explorado el espacio de soluciones en su totalidad, el conjunto del enjambre se encontrará volando alrededor de la zona con la mayor concentración de flores de todo el campo. En esta situación, las abejas, incapaces de encontrar posiciones alternativas mejores, son permanentemente atraídas hacia dicha posición como se muestra en la Figura 5.1.

------- Trayectoria
→ Memoria o nostalgia
→ Cooperación o conocimiento social



(a) (b)

Figura 5.1. Modelado del PSO utilizando como símil el movimiento de un enjambre de abejas sobre un campo con flores. (a) En su desplazamiento, las abejas son atraídas hacia las zonas de mayor concentración de flores encontradas personalmente por cada individuo (memoria) y por el conjunto del enjambre (cooperación). (b) Una vez que las abejas han sido atraídas a la zona con mayor concentración de flores, que equivale en términos de PSO a la convergencia hacia una solución global, éstas permanecen sobrevolando dicha zona con velocidades muy reducidas.

El comportamiento social que exhiben éste y otros organismos se puede entender como un método de optimización en el que el espacio de búsqueda se puede extender a N dimensiones del problema a optimizar, y donde cada partícula, se identifique como una posible solución potencial al problema, caracterizada por un vector velocidad y un vector posición, ambos en N dimensiones. El problema es reducible a establecer la ecuación que

dicte el movimiento de cada partícula de la población en el espacio $N-dimensional$ para mimetizar la inteligencia de estas comunidades y evitar a su vez caer en soluciones locales.

5.3 Operador Velocidad y parámetros del Algoritmo

En PSO la velocidad de partícula es el único operador disponible para controlar la evolución de la optimización. Ésta se define como, una población de I partículas donde cada partícula del enjambre se identifica con dos variables de estado inicializadas aleatoriamente dentro del espacio $N-dimensional$ que establece el problema a optimizar las cuales son:

- Un vector velocidad, $V_i = (v_{i1}, v_{i2}, ..., v_{iN})$, y
- Un vector de posición, $X_i = (x_{i1}, x_{i2}, ..., x_{iN})$.

Donde $i = 1, 2, ..., n$ corresponde a la i-ésima partícula y $N = 1, 2, ..., N$ corresponde a sus dimensiones. Este último corresponde a una solución potencial al problema de optimización. Los límites de los parámetros a optimizar, $x_n \in [x_{n,\min}, x_{n,\max}]$, conformando el espacio de búsqueda al cual debe restringirse el movimiento del enjambre.

En cada generación (iteración), cada una de las partículas es actualizada de acuerdo a dos valores importantes. El primero está asociado con la mejor solución históricamente visitada por ésta, $P_i = (p_{i1}, p_{i2}, ...p_{iN})$ y, el segundo, el cual conoce la posición de la mejor partícula o solución encontrada por todos sus congéneres, $G_i = (g_1, g_2, ..., g_N)$. El movimiento del enjambre se realiza en pasos temporales, que se traducen a nivel de algoritmo en iteraciones contiguas.

En cada iteración del método, k, cada una de las partículas de la población recorre el espacio de soluciones con una velocidad V_i hacia nuevas posiciones X_i, de acuerdo con su propia experiencia P_i, y con la experiencia aportada por el mejor de sus convecinos, "G". En las primeras versiones del algoritmo, esta formulación se reduce a las ecuaciones:

$$v_{in}(k+1) = v_{in}(k) + c_1 R_1(k) \cdot (p_{in}(k) - x_{in}(k)) + c_2 R_2(k) \cdot (g_n(k) - x_{in}(k)) \qquad (5.1)$$

$$x_{in}(k+1) = x_{in}(k) + v_{in}(k+1) \cdot \Delta t \qquad (5.2)$$

En la ecuación (5.1), $v_{in}(k)$ y $x_{in}(k)$ representan, respectivamente, la velocidad y posición en la iteración o instante de tiempo k de la partícula i en la dimensión n-ésima del espacio de búsqueda. Los factores c_1 y c_2 son los denominados constantes de aceleración cognitiva y social, que determinan en qué medida influyen sobre el movimiento de la partícula su propia memoria y la cooperación entre individuos, respectivamente. Los términos $R_1(k)$ y

$R_2(k)$ son dos números aleatorios independientes en el intervalo $[0,1]$, cuyo objetivo es emular el comportamiento estocástico y un tanto impredecible que exhibe la población del enjambre, y $\left(p_{in}(k)-x_{in}(k)\right)$ es la parte cognitiva que representa el aprendizaje de su propia experiencia y $\left(g_n(k)-x_{in}(k)\right)$ representa la parte social del aprendizaje de grupo. Después de calcular la nueva velocidad de la partícula i en la dimensión n, la nueva posición $x_{in}(k+1)$ se actualiza directamente de acuerdo con la ecuación (5.2), donde se asume que la velocidad se aplica durante un cierto período de tiempo Δt, típicamente de valor unitario. El proceso descrito se extiende al espacio $N-dimensional$, de forma que se van componiendo de manera interactiva nuevos vectores de posición x_i, utilizando, como en cualquier otro método de cómputo evolutivo, una función de fitness para ponderar la calidad de dicha solución parcial, actualizando los vectores p_i y g si se detectan resultados mejores.

El movimiento de los agentes sobre el espacio de soluciones y, en consecuencia, el rendimiento del algoritmo, está condicionado por el grado de contribución de los tres componentes de la velocidad (5.1) y denominados por este orden como momento, hábito o inercia para considerar la tendencia de la partícula; memoria, nostalgia o autoaprendizaje para incluir la experiencia de la propia partícula, y cooperación, conocimiento social, conocimiento de grupo o información compartida, para reflejar el intercambio de información y el comportamiento social como grupo.

Para acotar la velocidad de la partícula se especifica un valor máximo, $v_{\max}$, que restringe la velocidad en cada dimensión al intervalo $[-v_{\max}, v_{\max}]$. Si el valor de $v_{\max}$ es demasiado grande las partículas pueden sobrepasar e ignorar continuamente la zona con la solución global. En el extremo opuesto, si $v_{\max}$ toma valores extremadamente pequeños las partículas explorarán el espacio de soluciones muy lentamente y podrán quedar atrapadas alrededor de soluciones locales, incapaces de librarse de la base de atracción. Sin embargo, si no se introdujese el parámetro de control $v_{\max}$, el enjambre no convergería hacia un punto, sino que sufriría el fenómeno conocido como explosión del PSO, consistente en un comportamiento oscilatorio y creciente de la posición de las partículas.

El efecto $v_{\max}$ está fuertemente ligado con la naturaleza del problema a optimizar, y no existe ninguna regla que dicte las pautas a seguir para que el enjambre converja.

5.4 Control de la explosión

Con el objetivo de reducir el efecto de $v_{\max}$ y perfeccionar el control del alcance de la búsqueda sobre el espacio de soluciones, surgen versiones mejoradas del algoritmo que incorporan los conceptos de peso inercial [24] y el coeficiente de constricción.

5.4.1 Peso de inercia

Para cada instante en el tiempo (iteración) se incorpora el peso de inercia w ven la ecuación que determina la velocidad (5.1) de la partícula para balancear la búsqueda global y local

$$v_{in}(k+1) = w \cdot v_{in}(k) + c_1 R_1(k) \cdot \left(p_{in}(k) - x_{in}(k)\right) + c_2 R_2(k) \cdot \left(g_n(k) - x_{in}(k)\right) \qquad (5.3)$$

donde $w \in [0,1]$ favorece la convergencia de la partícula. Con este factor se controla la tendencia de la partícula a continuar en la dirección en la que se estaba moviendo y se regula la relación entre capacidad de exploración del espacio de búsqueda y habilidad de convergencia hacia las soluciones locales o globales, dictadas por el segundo y tercer sumando en (5.3).

El peso de inercia w en la ecuación (5.3) es considerado un parámetro crítico para la convergencia de PSO, ya que este se ocupa para controlar el impacto de la historia previa de las velocidades actuales teniendo como consecuencia regular la compensación entre las capacidades de exploración globales y locales del enjambre, ya que un peso inercial grande facilita el exploración global y un peso pequeño facilita la exploración local. Al poseer un valor conveniente, se reduce el número de iteraciones para localizar la solución óptima.

Los estudios realizados anteriormente en [13], revelan que valores altos para el coeficiente de inercia, factores de cognición y $v_{\max}(0,75 < w < 1 y 2 < c_1, c_2 < 4)$ favorecen el modo de exploración del PSO, mientras que valores bajos para estos parámetros $(0,4 < w < 0,75 y 0,1 < c_1 c_2 < 2)$ favorecen el modo de explotación del PSO.

5.4.2 Coeficiente de constricción

Una alternativa al modelo inercial lo constituye el denominado factor de constricción introducido por Clerc en [6].

Partiendo de la necesidad de encontrar un razonamiento a la necesidad del límite $v_{\max}$ y a la explosión del PSO, se iniciaron estudios para analizar la trayectoria que describen las partículas. La formulación de Ozcan y Mohan en [22] propone que las partículas se desplazan describiendo ondas sinusoidales, cambiando su frecuencia y amplitud aleatoriamente en su búsqueda de la posición óptima. En [6] se incluye una formulación mucho más exhaustiva, que deriva en la obtención de un modelo generalizado para PSO en el cual se puede controlar la explosión y la convergencia del algoritmo. El modelo propuesto por Clerc más utilizado es el descrito como constricción de tipo 1'', el cual modifica la ecuación 5.1 como:

$$v_{in}(k+1) = \chi \cdot \left(v_{in}(k) + c_1 R_1(k) \cdot \left(p_{in}(k) - x_{in}(k)\right) + c_2 R_2(k) \cdot \left(g_n(k) - x_{in}(k)\right)\right) \qquad (5.4)$$

Donde χ es el denominado coeficiente de constricción, el cual se expresa como:

$$\chi = \frac{2k}{\left|2 - c - \sqrt{c^2 - 4c}\right|},\ \mathrm{k} \in [0,1],\ c = c1 + c2,\ c > 4 \tag{5.5}$$

El coeficiente de constricción no está definido para $c < 4$. A medida que c toma valores por encima de 4.0, χ decrece paulatinamente, acentuando el efecto de amortiguamiento del movimiento. En este modelo de constricción típicamente se utilizan valores de $k = 1$ y $c = 4{,}1$ ($c_1 = c_2 = 2{,}05$), lo cual reporta un valor para el coeficiente de constricción de $\chi \approx 0{,}729$. Algebraicamente, esta configuración es equivalente a utilizar el modelo inercial de la ecuación (5.3) con $\omega = 0{,}729$ y $c_1 = c_2 = 1{,}49445$.

La formulación de PSO se reduce a caracterizar el movimiento de las partículas en base a un operador velocidad que debe aunar exploración y convergencia, descomponiendo para ello la velocidad en tres componentes en un intento por sintetizar el comportamiento social descrito en la Figura 5, tal y como se muestra en la Figura 5.2.

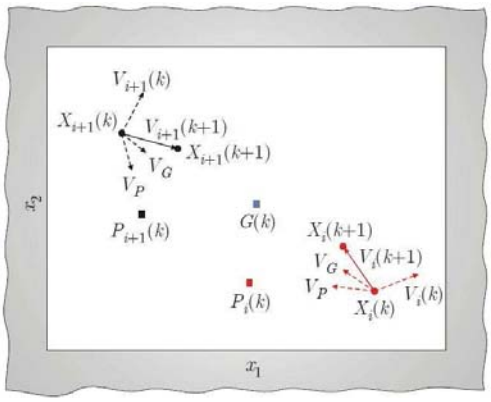


Figura 5.2. Componentes del vector velocidad. Movimiento sobre un espacio bidimensional de las partículas i e i+1 en la iteración k+1.

A modo de resumen, en la Tabla 5.1 se exponen las configuraciones típicas de PSO más utilizadas.

TABLA 5.1. PARÁMETROS ASOCIADOS CON CONFIGURACIONES TÍPICAS DEL PSO

Modelo del PSO	**Parámetros básicos del algoritmo.**
Peso inercial.	ω=0.9-0.4 (linealmente decreciente), c_1=c_2=2.0
Peso inercial.	$\omega = 0.9 - 0.4$ (linealmente decreciente), $c_1 = c_2 = 1.49445$
Peso inercial.	$\omega = 0.5 + rnd/2.0$, $c_1 = c_2 = 1.49445$
	Donde rnd representa a un número aleatorio con distribución uniforme $U[0,1]$.
Peso inercial.	$\omega = 0.95 - 0.2$ (linealmente decreciente), $c_1 = c_2 = 2.0$
Peso inercial.	$\omega = 0.4$, $c_1 = c_2 = 2.0$
Factor de Constricción.	$\chi = 0.729$, $\varphi_1 = \varphi_2 = 2.05$, κ=1
	$\chi = 0.729$, $\varphi_1 = 2.8$, $\varphi_2 = 1.3$, κ=1
	$v_{\max} \le \lvert x_{n,\max}, x_{n,\min} \rvert, I \in [10,50]$

5.5 Límites en el espacio N-dimensional

En aplicaciones en el campo de la ingeniería, normalmente se precisa limitar el espacio de soluciones a la región donde están definidas las variables a optimizar. Sin embargo, la naturaleza del PSO, aún incluyendo las restricciones impuestas por medio de la velocidad máxima, del peso inercial y del factor de constricción, no asegura que las partículas se confinen dentro del espacio de soluciones al aplicar la ecuación (5.2). Para conseguirlo se introducen diferentes mecanismos que, dimensión a dimensión, controlan la posición de destino de la partícula y en caso de exceder los límites modifican los vectores de posición y/o velocidad de la misma.

En la figura mostrada a continuación, se encuentra los cuatro tipos de barreras de contención que se han investigado. Básicamente, la ***pared absorbente*** lleva la partícula al límite de la dimensión y anula su velocidad. La ***pared reflectante*** invierte el signo de la velocidad en la dimensión excedida y refleja la componente del vector de posición en dicha dimensión. La ***pared invisible*** no restringe el movimiento de las partículas, pero el fitness de aquéllas que exceden los límites no se calcula. Por último, la ***pared frontera*** es similar a la pared absorbente, con la única diferencia de que ahora el vector velocidad no se modifica.

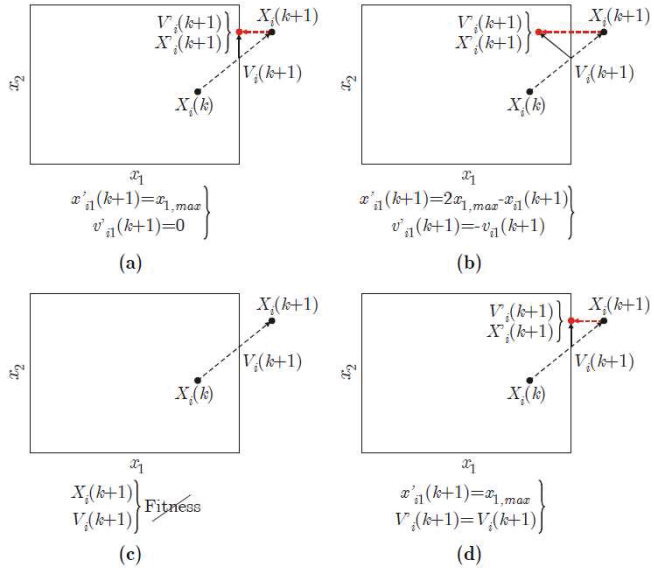


Figura 5.3. Límites impuestos para restringir el espacio de soluciones al rango dinámico de las variables. Ejemplo particularizado para el espacio bidimensional. (a) Pared absorbente. (b) Pared reflectante. (c) Pared invisible. (d) Pared frontera. **[23]**

5.6 Exploración y Explotación en PSO

Se define como exploración, a la búsqueda de la solución del problema en el espacio de búsqueda por rango amplio, es decir, que las partículas vuelen por diferentes zonas del espacio de búsqueda privilegiando la diversificación del enjambre. Explotación se define a la búsqueda de la solución del problema encontrado un rango (sub-espacio) dentro del espacio de búsqueda, en donde el enjambre trate de tomar todos los valores posibles que comprenden dicho rango privilegiando la convergencia del enjambre [9].

En relación a lo expresado en el párrafo anterior, se puede concluir que lo ideal es que las partículas (o población) representen soluciones de diferentes zonas del espacio de búsqueda, es decir, lo más diversificado posible. Este concepto se conoce como diversificación de la población, y en términos del algoritmo, se refiere a que las partículas sobrevuelen diferentes zonas del espacio de solución antes de que se comprometan con alguna zona. Por otro lado, una de las características que debe tener todo algoritmo de optimización, es su capacidad de converger a una solución óptima, conocida como convergencia de la población, lo cual se refiere a que las partículas, una vez conglomeradas en una zona prometedora, sean capaces de mejorar la solución que hayan encontrado, es decir, que logren encontrar y converger a un óptimo cada vez mejor [10] [8].

El algoritmo PSO se presenta en la tabla 5.2.

TABLA 5.2. ALGORITMO PSO TRADICIONAL.

Paso 1: configurar el tamaño del enjambre n e inicializar las posiciones y velocidades de las partículas en un espacio N-dimencional.
Paso 2: Para cada partícula, evaluar su fitness de acuerdo a algún criterio de optimización.
Paso 3: Comparar el fitness actual de cada partícula con el fitness de su mejor posición personal encontrada hasta el momento. Si el fitness actual resulta ser mejor, entonces se actualiza la mejor posición personal de la partícula p_{in} al valor de la posición actual x_{in}.
Paso 4: Comparar el fitness actual de cada partícula con el mejor fitness encontrado en el enjambre. Si el fitness actual resulta ser mejor, entonces se actualiza la mejor posición global g_n al valor de la posición actual x_{in}.
Paso 5: Ajustar las Velocidades y Posiciones de las partículas de acuerdo a la ecuación de PTSP ocupada. Verificar si las partículas sobrepasan la velocidad máxima definida.
Paso 6: Verificar si se cumple algún criterio de término (máximo número de iteraciones, valor de fitness alcanzado, número de intentos de mejoramiento de la solución), sino ir al Paso 2.

El éxito del algoritmo, radica en su capacidad de ajustar las posiciones de las partículas en un área del espacio de soluciones prometedora, de acuerdo a una función objetivo que se desea minimizar o maximizar. Es decir, la función objetivo evalúa la aptitud o fitness de una partícula, y se utiliza para guiar la búsqueda durante el proceso de optimización.

Las condiciones de término comúnmente usadas para finalizar el proceso de optimización son:

- **Número máximo de iteraciones:** El proceso termina después de alcanzar un número fijo de iteraciones.

- **Número de iteraciones sin mejoras:** El proceso de iteración termina después que se alcanza un número fijo de iteraciones en la que no se han obtenido alguna mejora en términos de solución.

- **Error mínimo de la función objetivo:** El error entre el valor obtenido de la función objetivo y el mejor valor de eficacia (fitness) es menor que un umbral prefijo esperado.

El mal ajuste de los parámetros puede provocar que el PSO converja a un óptimo en pocas iteraciones, o a un buen óptimo en muchas iteraciones. A menudo, el PSO puede encontrar un mal óptimo en pocas iteraciones (conocido como convergencia prematura), o a un mal óptimo en muchas iteraciones.

5.7 Esquemas de PSO

Se establece una clasificación de los esquemas clásicos de PSO atendiendo principalmente a dos características del enjambre (swarm):

- La forma en la que se sincroniza la transmisión de información entre los agentes de la población y la topología que se confiere a la población, es decir, el modo en el que se transmite la información entre cada partícula y
- Sus convecinos.

Atendiendo a esta clasificación surgen las versiones con actualizaciones síncrona y asíncrona, y las topologías *global y local.*

5.7.1 Topologías de vecindad global y local

En PSO los individuos mejoran sus aptitudes imitando los comportamientos y tendencias que encuentran en los mejores congéneres de la población, por lo que tiene una trascendencia vital el establecer la vecindad del individuo (que otros individuos influencien a éste) en el rendimiento del algoritmo. Dependiendo de la topología que adquiera la población, la transmisión de la información entre individuos se puede acelerar o ralentizar, lo cual está íntimamente relacionado con la velocidad de convergencia y con la capacidad del algoritmo para escapar de soluciones locales.

Éste puede ser descrito, de manera general, como una población de vectores cuyas trayectorias oscilan alrededor de una región que está definida por el mejor éxito personal de cada individuo y el éxito de alguna otra partícula. Se han usado varios métodos para identificar otra partícula que influya en el individuo. Los autores de [13], Eberhart y Kennedy, llamaron a los dos métodos básicos como “modelo del óptimo global” y “modelo del óptimo local”. En el modelo del óptimo local, las partículas tienen sola la información de sus vecinos más cercanos del enjambre, mientras que el modelo del óptimo global las partículas consideran la información de todo el enjambre.

La topología más extendida es la topología de red global, en la cual todos los individuos están interrelacionados y tienen acceso inmediato a los hallazgos de sus congéneres. Sin embargo, esta estructura social es vulnerable a soluciones locales, dado que dependiendo de la distribución puntual de las partículas sobre el espacio de soluciones, una de ellas, apuntando a una solución local, puede llegar a dominar al resto.

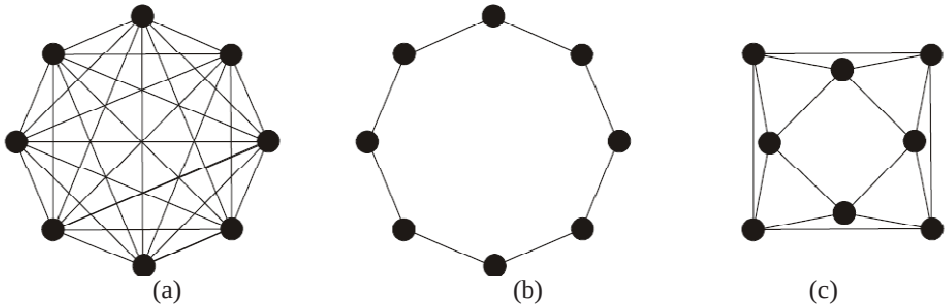


Figura 5.4. Topologías de la población. (a) Global. (b) Local con $N_v = 2$. (c) Local con $N_v = 4$. Donde $N_v =$ vecinos adyacentes.

Es obvio que la información extremo a extremo entre dos partículas, que se encuentren alejadas entre sí, fluye más lentamente, lo cual justifica que a cambio de una mejor exploración la versión local sea más lenta que su homónima global. A medida que aumenta el número de vecinos, tal y como sucede al pasar de la configuración de la Figura (b) a la Figura (c), el aislamiento de las partículas disminuye, aumentando la cooperación y, en consecuencia acelerado la convergencia. Si el valor de N_v es excesivamente alto el enjambre imita el comportamiento de la topología global.

Los autores de [28] declaran que el modelo de óptimo global converge rápidamente hacia las soluciones del problema, pero tiene la debilidad de quedar atrapado en óptimos locales, mientras el modelo de óptimo local tiene la desventaja de converger lentamente hacia la solución del problema. Por ello, recomiendan usar el modelo de óptimo global fuerte para funciones unimodales, mientras recomiendan a un modelo local variable para funciones multimodales.

5.7.2 PSO con actualizaciones síncronas y asíncronas de la población

Al atravesar el espacio N-dimensional, las partículas actualizan sus vectores de posición y velocidad de acuerdo con su hábito, su memoria y el conocimiento social.
En función del instante dentro del proceso iterativo en el que se realiza la actualización de la memoria de cada partícula y el conocimiento social del grupo, se puede distinguir entre PSO con actualizaciones síncronas y asíncronas de la población.

En el modelo síncrono todas las partículas se mueven en paralelo. En cada iteración se evalúa el fitness de todas las partículas, se actualiza su memoria p_{in} y el conocimiento social g, y a continuación la población se desplaza hacia un nuevo punto, tomando como referencia esta dupla de información. Es decir, todas las partículas comparten la misma información acerca de la mejor solución de partida g.

Por el contrario, en el PSO asíncrono cada partícula aprovecha al desplazarse la información actualizada por sus inmediatos predecesores. Es decir, en cada iteración k, la partícula i-ésima se desplaza hacia un nuevo punto $x_i(k+1)$ utilizando la información de los vectores $p_{in}(k)$ y g, actualizados por las $i-1$ partículas previas. Posteriormente, la partícula evalúa la calidad del nuevo punto y actualiza, si procede, las variables $p_{in}(k+1)$ y g. Esta información se transmite a las restantes partículas. Al actualizar la información partícula a partícula, el modelo asíncrono acelera la optimización, aunque la naturaleza del modelo síncrono lo hace susceptible de ser ejecutado en paralelo, sobre múltiples procesadores.

5.8 Descripción del algoritmo PSO.

A continuación se presenta el diagrama de flujo correspondiente al algoritmo PSO global con actualizaciones asíncronas.

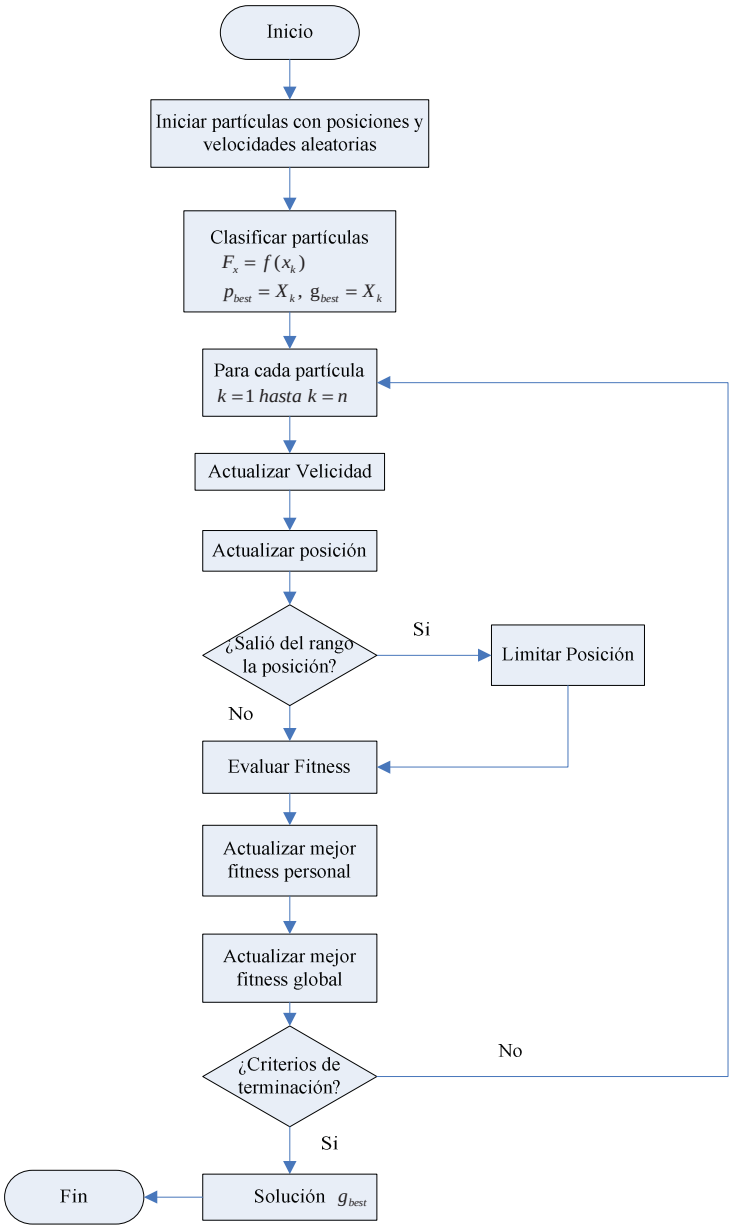


Figura 5.5. Diagrama de flujo de algoritmo PSO con actualizaciones asíncronas.

Como se muestra en el diagrama de flujo, ésta partícula parte con posiciones y velocidades aleatorias para cada partícula que nos dará una configuración inicial factible (dentro del espacio de solución). Se actualiza la velocidad y posición de cada partícula tomando en cuenta que ésta no pase del rango dado previamente (este genera una nueva solución), para

luego evaluar el fitness y revisar si es un fitness mejor al que se tiene en forma local y global.

Dentro de los criterios de detención del algoritmo puede estar el número de iteraciones y/o la aproximación al resultado deseado que se quiere encontrar.

5.9 Pseudocódigo PSO

A continuación se muestra el pseudocódigo recorre todas las partículas haciendo una evaluación de la función fitness (costo de la partícula x_i) viendo si se encontró uno mejor valor local para su reemplazo. Si también la función de evaluación es mejor que la histórica, actualiza el valor actual como mejor valor. Esto se repite hasta un total de iteraciones dadas o encuentre un mejor valor fitness final al propuesto.

1. Inicializar las partículas y las velocidades con valores aleatorios.

2. Iniciar Fitness local y global que son posibles.

Hacer
3. Desde $i=1$ hasta el total de partículas
4. Si $\left(f(x_i) < f(p_i)\right)$ entonces //Evaluación función fitness
6 Desde $n=1$ hasta dimensiones
7. $p_i = x_i$ // p_{in} mejor encontrado localmente
8. Fin desde
9. Fin Si
10.
11. Si $\left(f(x_i) < f(g_i)\right)$ entonces //g es el índice del vecino
// con mejor desempeño
12. Desde $n=1$ hasta dimensiones
13. $g_i = x_i$ // p_{in} mejor encontrado globalmente
14. Fin desde
15. Fin Si

16. $v_{in}(k+1) = w \cdot v_{in}(k) + c_1 R_1(k) \cdot \left(p_{in}(k) - x_{in}(k)\right) + c_2 R_2(k) \cdot \left(g_n(k) - x_{in}(k)\right)$
17. $v_{in} \in (-V_{\max}, +V_{\max})$
18. $x_{in}(k+1) = x_{in}(k) + v_{in}(k+1) \cdot \Delta t$

19. Fin Desde
20. Mientras (no_concluyan_iteraciones o no_se_encuentre_valor_fitness)

CAPÍTULO 6

SIMULATED ANNEALING (SA)

Simulated Annealing (recocido simulado) es una metaheurística probabilística genérica para problemas de optimización global de las matemáticas aplicadas, es decir, la localización de una buena aproximación al mínimo global de una función de búsqueda en un gran espacio. A menudo se usa cuando el espacio de búsqueda es discreto (por ejemplo, todos los tours que visitan un determinado conjunto de ciudades). Para ciertos problemas, Simulated Annealing puede ser más eficaz que la enumeración exhaustiva ya que este tiene como objetivo de encontrar una solución aceptable, más que la mejor solución posible.

Esta es una heurística para la solución de problemas de optimización, que en la práctica funciona de las siguiente manera: consiste en recalentar un metal una vez este ha sido moldeado y enfriarlo lentamente, este proceso ayuda a que los cristales dentro del metal se reorganicen y encuentren un estado de equilibrio de energía mínima.

La manera de utilizar esta heurística para solucionar problemas de optimización en la teoría es adaptar la ecuación para el recocido a la programación, de esta manera se recorre el espacio de soluciones del problema de una manera veloz (excitada) y, mientras se reduce la temperatura sintética, se reduce el grado de excitación con el que se recorre el espacio de soluciones acercándose cada vez más a la solución, esta solución generalmente no es un optimo global sino uno muy cercano a este. El beneficio de utilizar SA para hallar el mínimo óptimo de un problema es que si el espacio de soluciones es demasiado grande, este algoritmo lo recorre rápidamente.

6.1 Analogía Física

El método del recocido se utiliza en la industria para obtener materiales más resistentes, o más cristalinos, en general, para mejorar las cualidades de un material.

El proceso consiste en “derretir” el material (calentarlo a muy alta temperatura). En esa situación, los átomos adquieren una distribución “azarosa” dentro de la estructura del material, y la energía del sistema es máxima. Luego se hace descender la temperatura muy lentamente por etapas, dejando que en cada una de esas etapas los átomos queden en equilibrio, es decir que los átomos alcancen una configuración óptima para esa temperatura. Al final del proceso, los átomos forman una estructura cristalina altamente regular, el material alcanza así una máxima resistencia y la energía del sistema es mínima.

Experimentalmente se comprueba que si la temperatura se hace descender bruscamente o no se espera suficiente tiempo en cada etapa, al final la estructura del material no es la óptima.

La rama de la Física llamada Mecánica Estadística se ha encargado de desarrollar una serie de métodos para estudiar el comportamiento de grandes cantidades de átomos de un sistema. Debido a que en promedio, en un sistema hay 10^{23} átomos por cm^3, solamente puede estudiarse el comportamiento más probable del sistema en equilibrio a una temperatura dada. La experimentación mostró que los átomos de un sistema en un proceso de recocido se comportan según el factor de probabilidad de Boltzman.

En 1953 Metrópolis modeló el proceso de recocido, en cada paso del algoritmo se le da al átomo un desplazamiento azaroso y se mide el cambio de energía ΔE.

Las leyes de la termodinámica dicen que a una temperatura t la probabilidad de un incremento energético de magnitud ΔE se puede aproximar por:

$$P[\Delta E] = e^{-\Delta E / kt} \tag{6.1}$$

donde t es la temperatura del sistema y k es la constante de Boltzman. En este algoritmo se genera una perturbación aleatoria en el sistema y se calcula los cambios energéticos resultantes si:

- $\Delta E \leq 0$, se acepta el desplazamiento.
- $\Delta E > 0$, se acepta el desplazamiento con probabilidad $e^{-\Delta E / kt}$

Esto significa que si la temperatura es alta no interesa tomar decisiones erradas, pero mientras la temperatura disminuye somos forzados a acomodarnos en la configuración de menor energía que se encuentre en nuestro vecindario.

Otra forma de decir es:

$$P_{aceptacion} = \begin{cases} 1 & \text{, si f(j) } \leq \text{ f(i)} \\ e^{-\left(\frac{f(j)-f(i)}{T}\right)} & \text{, si f(j) > f(i)} \end{cases} \tag{6.2}$$

El proceso se repite durante un número predefinido de iteraciones en series decrecientes de temperatura, hasta que el sistema este “frío”.

A principios de los años 80, publicaciones independientes de Kirkpatrick (1983) sobre diseños de circuitos VLSI y Cerny (1985) para el TSP, mostraron cómo este proceso podría ser aplicado a problemas de optimización, asociando conceptos claves del proceso original de simulación con elementos de optimización combinatorial según se indica en la tabla

TABLA 6.1. ASOCIACIÓN DE CONCEPTOS CLAVES DEL PROCESO ORIGINAL DE SIMULACIÓN, CON ELEMENTOS DE OPTIMIZACIÓN COMBINATORIAL. [20]

Simulación termodinámica	Optimización combinatorial.
Estados del Sistema.	Soluciones Factibles.
Energía.	Costos.
Cambio de Estado	Solución en el Entorno.
Temperatura.	Parámetro de Control.
Estado Congelado.	Solución Heurística.

La idea básica es, partir de una solución inicial seleccionando al azar una solución vecina. Si la solución vecina es mejor se adopta como la nueva solución; si no lo es, se acepta como nueva solución una que tenga una probabilidad de decrecer a medida que el algoritmo avanza. La idea es aceptar soluciones de menor calidad para que el algoritmo le permita salir de óptimos locales como se muestra en la figura.

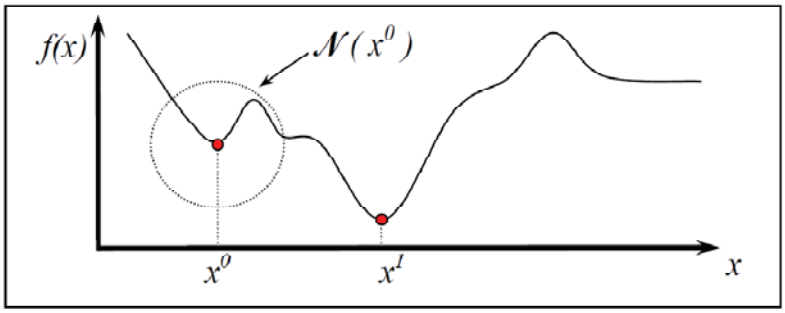


Figura 6.1. Figura que muestra óptimo local en la posición X^0 y óptimo global que se encuentra en la posición X^1.

6.2 Definición Simulated Annealing (SA) [20]

Sea S el espacio de soluciones (factibles). Sea $i \in S$ una solución del problema. Sea $f(i)$ la función objetivo definido para el espacio de soluciones. El objetivo es encontrar un mínimo global i^* tal, que se cumpla:

$$i^* \in S, f(i) \geq f(i^*) \forall i \in S \tag{6.3}$$

Se tiene que $N(i)$ es el vecindario de $i \in S$; de lo anterior se puede decir que por cada solución $i \in S$ tenemos que existen soluciones vecinas las que pueden ser alcanzadas en una sola iteración del algoritmo.

SA inicia con una solución inicial $i \in S$. Una vez que tenemos la solución inicial, se obtienen las soluciones que pertenecen a su vecindario, es decir, $j \in N(i)$. Luego,

utilizando la probabilidad de que j sea seleccionado dado que estamos en i, la que se define como:

$$q_{ij} \geq 0, \sum_{j \in N(i)} q_{ij} = 1 \tag{6.4}$$

Se selecciona una solución vecina que se comparará con la solución actual. Una vez que estas soluciones son comparadas, la probabilidad de aceptar una solución que mejora a la actual es 1. En el caso que no sea una solución que mejore el resultado, se debe calcular una probabilidad de aceptación mediante la fórmula:

$$P(j) = e^{-\left(\frac{f(j)-f(i)}{t_k}\right)} \tag{6.5}$$

Donde t_k es el parámetro de temperatura en la iteración k, se puede definir de la siguiente manera:

$$t_k > 0, \forall k \quad y \quad \lim_{k->\infty} t_k = 0 \tag{6.6}$$

Otro parámetro importante es el factor de reducción de la temperatura r que varía entre: $0 < r < 1$, con el cual se reducirá el parámetro de temperatura t_k en cada iteración.

Este mecanismo de aceptación es el elemento principal de SA. Si la temperatura se reduce lo suficientemente lento, y además se dispone de tiempo $t \to \infty$, se puede llegar a la solución óptima del problema. En la mayoría de los casos, esto es prácticamente imposible, por lo que se debe conformar con una solución que se aproxime de la mejor manera.

6.3 Datos iniciales y parámetros

6.3.1 Temperatura inicial (T_0)

La temperatura inicial T_0 debe ser una temperatura que permita casi (o todo) movimiento, es decir que la probabilidad de pasar del estado i al j (en $N(i)$) sea muy alta, sin importar la diferencia $f(j) - f(i)$. Esto es que el sistema tenga un alto grado de libertad. En problemas como TSP, donde la entrada son los nodos de un grafo y las soluciones posibles son distintas formas de recorrer estos nodos, puede tomarse T_0 proporcional a la raíz cuadrada de la cantidad de nodos. En general se toma un valor T_0 que se cree suficientemente alto y se observa la primera etapa para verificar que el sistema tenga un grado de libertad y en función de esta observación se ajusta T_0.

6.3.2 Solución inicial (i_0)

En todas las versiones, el sistema debe ser "derretido" antes de implementar el algoritmo. Esto es que la solución factible inicial que llamamos i_0 debería ser una solución tomada al azar del conjunto de soluciones factibles. En algunos problemas esto puede hacerse utilizando números pseudo-aleatorios provistos por uno. Pero en muchos casos ya es problemático encontrar una solución, por lo que es imposible tomar una al azar. En estos casos se implementa un algoritmo "greedy" tipo búsqueda local para buscar una solución factible y se toma esta como i_0 (ejemplo de esto es el TSP).

6.3.3 Criterio de cambio de la temperatura

Se usan dos parámetros:

K : Cantidad de iteraciones que estamos dispuestos a hacer en cada etapa (equivalente a la cantidad de tiempo que vamos a esperar a que el sistema alcance su equilibrio térmico para una dada temperatura T).

A : Cantidad de aceptaciones que se permiten hacer en cada etapa.

A medida que la temperatura disminuye se supone que al sistema le resulta más difícil alcanzar un equilibrio, porque es más dificultoso el movimiento, entonces hay que esperar más tiempo, esto se traduce en aumentar K.

6.3.4 Parámetro de aumento de K

En general se utiliza un parámetro de congelamiento (frozen: FRZN). Como a medida que disminuye la temperatura, aumenta el parámetro K y A permanece constante, la proporción A/K se hace pequeña. Asumimos que si $A/K < FRZN$ el sistema está congelado (la cantidad de aceptaciones respecto de la cantidad de iteraciones es muy chica, esto da la idea de que cambiar de configuración es muy difícil).

6.3.5 Factor de enfriamiento [18]

El posible calendario de refrigeración en el presente documento es:

$$T = (\alpha)^i T_0 + T_\theta \tag{6.7}$$

Donde el coeficiente de refrigeración α es una constante aleatoria entre 0 y 1, "i" es el numero de iteraciones operadas hasta la fecha, T_0 es la temperatura inicial y T_θ es el valor de la temperatura más baja.

6.3.6 Descripción del algoritmo SA

A continuación se presenta el diagrama de flujo correspondiente al algoritmo SA general empleado para un problema de minimización.

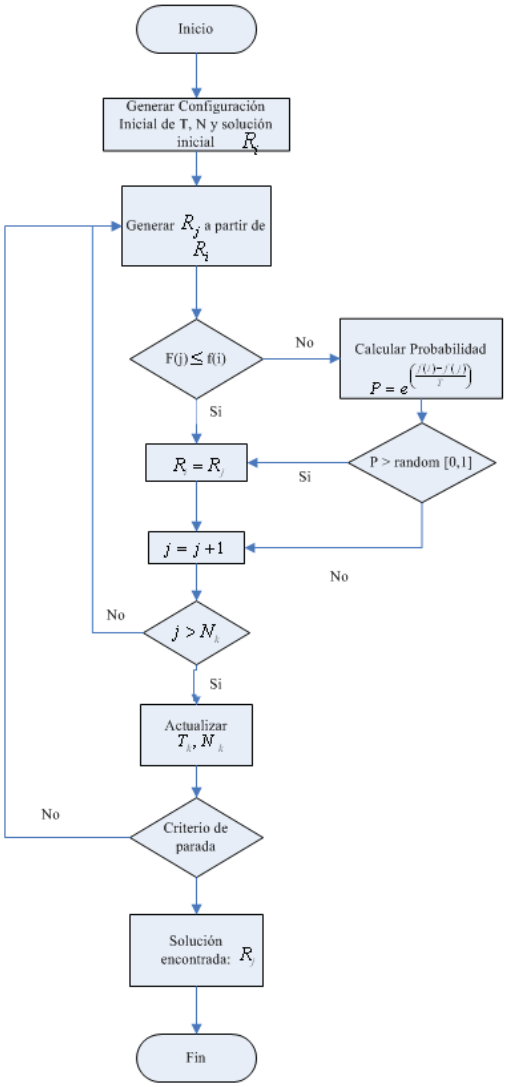


Figura 6.2. Diagrama de flujo de algoritmo SA.

Como se muestra en el diagrama de flujo, ésta parte con una temperatura inicial, un número de iteraciones y una configuración inicial factible (dentro del espacio de solución). A partir de este último es que se genera una nueva solución las cuales se comparan (solución inicial

y nueva solución) para ver si es aceptada esta última; si es así es tomada como nueva solución factible reemplazando a la anterior. Esto se hace durante todo el proceso.

Dentro de los criterios de detención del algoritmo puede estar el número de iteraciones y/o la aproximación al resultado deseado que se quiera encontrar.

6.4 Pseudo-código Algoritmo SA

El siguiente código representa una versión general del algoritmo de recocido simulado:

1. Seleccionar una solución inicial $i \in S$.
2. Seleccionar un factor de temperatura t_k inicial, y su respectivo factor de reducción r.
3. Repetir hasta que $t_k \approx 0$, o se encuentre un óptimo local.
4. Generar vecinos $N(i)$
5. Hacer hasta aceptar algún vecino o ya no queden más.
6. Seleccionar un vecino j utilizando números aleatorios para la probabilidad.
7. Calcular $\Delta = f(j) - f(i)$
8. Si $\Delta \leq 0$
9. $I = j$
10. Si $\Delta \geq 0$
11. Obtener numero aleatorio entre 0 y 1, y si $n > e^{\left(\frac{-\Delta}{t_k}\right)}$ asignar
12. $I = j$
13. Asignar a $t_k = r * t_k$
14. Entregar la mejor solución encontrada.

CAPÍTULO 7

MODELO Y DISEÑO DE LA SOLUCIÓN

7.1 Modelo de los algoritmos

El modelo propuesto para el proyecto de título consta de 3 partes que corresponden a la función PTSP y; de las metaheurísticas PSO y SA.

- La función de PTSP a utilizar corresponde a distribuciones de probabilidad homogénea, donde $p_i = p$ y $q_i = q$ para cada cliente i. Esta función está dada por:

$$E[L(\tau)] = \sum_{i=1}^{n} \sum_{r=1}^{n-1} p^2 q^{r-1} d_{\tau(i)\tau(i+r)}$$

  La elección de esta función en particular de PTSP es para comparar los resultados obtenidos con los resultados del paper:

  - "Solving the Probabilistic TSP with Ant Colony Optimization"

  La distancia euclidiana que se usará para el cálculo de cada par de nodos es:

$$d_{ij} = \sqrt{\left(x_i - x_j\right)^2 + \left(y_i - y_j\right)^2} \qquad (7.1)$$

- El algoritmo PSO utilizado es PSO con peso inercial ω, que es mostrado en el capítulo 5.

- El calendario de refrigeración utilizado en SA es el sacado del paper "*Particle Swarm Optimization with Simulated Annealing for TSP*". Esto hace que la temperatura disminuya en cada iteración afectando a su vez la aceptación de la nueva ruta.

7.2 Modelo de Enjambre de Partículas y Simulated Annealing

El modelo de enjambre de partículas con Simulated Anneling preliminar consiste en aplicar este modelo a un PTSP en el cual encuentre la ruta mínima del recorrido.

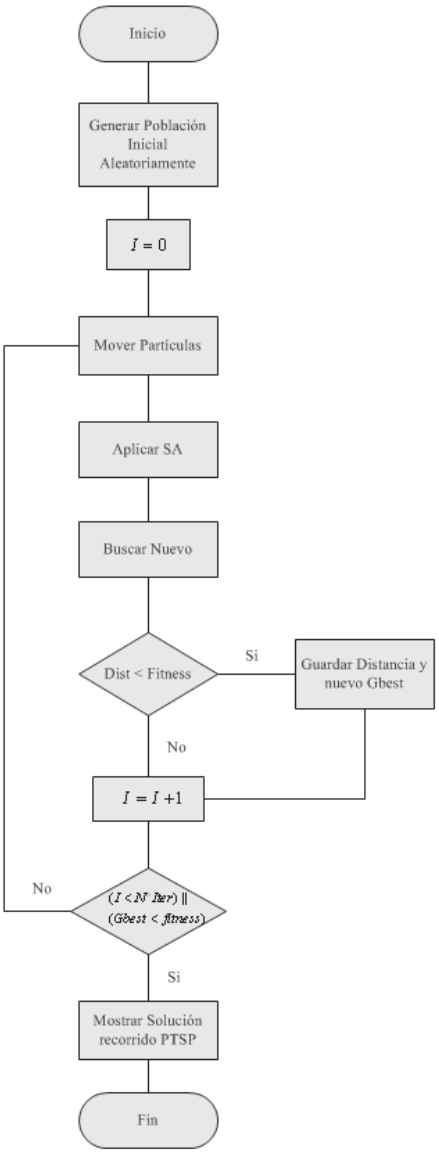


Figura 7.1. Diagrama de Flujo de modelo PSO con SA para resolver PTSP.

Del diagrama de flujo anterior se pueden desprender una serie de procesos donde destacan: Generar la población inicial de partículas aleatoriamente, evaluar el fitness de la población de enjambre de partículas, etc. Dichos procesos se describen más adelante del capítulo.

7.3 Distancia entre nodos

Ésta será representada por una matriz cuadrada de nxn para guardar la distancia de los nodos. Se consideran las distancias como simétricas.

TABLA 7.1. REPRESENTACIÓN DE DISTANCIAS SIMÉTRICAS ENTRE NODOS DE **MATRIZ** nxn**.**

	1	**2**	**…**	**N**
1	0	D_1	…	D_n
2	D_1	0	…	D_2
…	…	…	0	…
N	D_n	D_2	…	0

En base al marco teórico mostrado anteriormente, se dará un diseño de la solución a evaluar para el problema del vendedor viajero probabilístico donde se tomarán en cuenta los siguientes parámetros a la hora de llevar a cabo la implementación.

7.4 Inicialización de individuos

Los individuos se inicializarán con valores aleatorios, tanto para la posición como para la velocidad. Dichos valores, a pesar de ser aleatorios, se deben encontrar dentro del rango del espacio de búsqueda. Este tipo de inicialización, hace que se pueda apreciar más fehacientemente el comportamiento de PSO.

Éstas serán definidas previamente en la cual se creara una matriz llamada partícula de N _partícula x N _dimensión donde:

N_dimensión: Número de ciudades a evaluar.

Cada fila de esta matriz corresponde a la visión que tiene la partícula de cada ciudad y los vecinos que posee.

7.5 Construcción de matriz de partículas inicial

Cada partícula se inicializará de forma aleatoria en donde cada posición de la fila corresponde a una ciudad. Se debe tomar en cuenta que ésta tiene que ser una ciudad única en esa fila.

En la matriz partícula estará representada cada ciudad con un número dado al comienzo.

La velocidad y posición de cada partícula se tomará al comienzo como un valor aleatorio entre 0 y el número de ciudades menos uno.

La mejor posición corresponderá a la partícula que tenga la menor distancia encontrada por el enjambre.

Por ser la primera generación de partículas, la mejor posición será igual a la posición de la partícula.

7.6 Construcción de nueva partícula

Si, "S_1" es la partícula que se va a comunicar con la partícula vecina "S_2". S1 se constituirá como una nueva partícula a través de la combinación de S_1 y S_2.

Para entender la manipulación y creación de la nueva partícula es que se muestra a continuación un ejemplo de cómo funciona el método para la creación de la nueva partícula

Si el número total de ciudades es de 10, y las partículas S_1 y S_2 son los siguientes:

S_1	***0***	***3***	***5***	***9***	***6***	***4***	***1***	***2***	***8***	***7***
S_2	**7**	**9**	**6**	**4**	**8**	**2**	**1**	**3**	**5**	**0**

Para la primera generación de las partículas de forma aleatoria se tendría que proceder de la siguiente manera:

Primera manipulación: Seleccionar la posición de la ciudad (best) para producir el movimiento de ésta (suponer que se ha seleccionado la ciudad 4) y añadirla a S_1.

Los vecinos de la ciudad 4 son las ciudades 6, 8 y 1

S_1	***0***	***3***	***5***	***9***	***6***	***4***	***1***	***2***	***8***	***7***
S_2	**7**	**9**	**6**	**4**	**8**	**2**	**1**	**3**	**5**	**0**

Siendo que las ciudades que se encuentran más cerca de la ciudad 4 son las ciudades 6 y 8, las que son elegidas para ser los vecinos sobre la izquierda y la derecha en la nueva partícula S_1.

Para hacer la elección de los vecinos se utiliza el algoritmo burbuja para encontrar la menor distancia entre la ciudad insertada y sus vecinos.

S_p					6	4	8			

Segunda manipulación: al considerar el vecino de la derecha de la ciudad 4, los vecinos de la ciudad 8 son las ciudades 2, 4 y 7 en las partículas S_1 y S_2

S_1	0	3	5	9	6	4	1	2	8	7
S_2	7	9	6	4	8	2	1	3	5	0

Pero la ciudad 4 en S_1 ya se encuentra seleccionada por lo que se elije la ciudad 2 como acompañante derecho de la ciudad 8 (la ciudad 2 es elegida por ser la que posee menor distancia entre la ciudad 8 y sus vecinos). La ciudad 2 será insertada de forma que la distancia sea mínima entre su vecino anterior y el vecino anterior a éste; es decir si la distancia entre los vecinos a insertar:

4	8	2

Es menor a:

4	2	8

Será insertado de la forma:

S_p					6	4	8	2		

Su inserción será:

S_p					6	4	2	8		

Ahora supongamos que la menor distancia se encuentra al insertar el vecino de la forma:

S_p					6	4	8	2		

Con esta indicación termina la segunda manipulación.

Tercera manipulación: Si se considera el vecino izquierdo de la ciudad 4, los vecinos de la ciudad 6 son 9 y 4 en las partículas S_1 y S_2

S_1	0	3	5	9	6	4	1	2	8	7
S_2	7	9	6	4	8	2	1	3	5	0

La ciudad 4 se supone bien colocada y se encuentra ya en el vector, solamente la ciudad 9 puede ser el vecino de la izquierda de la ciudad 6. Al igual que en la segunda manipulación se procede a ver si es más conveniente, en término de distancias, colocar:

9	6	4

Ó

6	9	4

Supongamos que la menor distancia es al insertar la cuidad de la forma indicada.

S_p				9	6	4	8	2		

Con esto termina la tercera manipulación.

Cuarta manipulación: Considere el vecino derecho de la ciudad 2, los vecinos de la ciudad 2 son 8 y 1 en S_1 y S_2, la ciudad 8 se supone bien colocada y se encuentra ya en el vector, solamente la ciudad 1 puede ser el vecino de la derecha de la ciudad 2 y, al igual que en la manipulación anterior se procede a la inserción tratando de obtener la menor distancia posible, quedando de la forma siguiente.

S_p				9	6	4	8	2	1	

Quinta manipulación: Si se considera el vecino izquierdo de la ciudad 9, los vecinos de éstas son 5, 6 y 7 en S_1 y S_2, la ciudad 6 se supone bien colocada y se encuentra ya en el vector, por lo que se elige entre la ciudad 5 y 7, la ciudad 7 posee la menos densidad, quedando así:

S_p			7	9	6	4	8	2	1	

Sexta manipulación: considere el vecino derecho de la ciudad 1. El único vecino que se encuentra sin colocar, es la ciudad 3 por lo que se inserta en el vector.

S_p			7	9	6	4	8	2	1	3

Séptima manipulación: Considere el vecino izquierdo de la ciudad 7. Todos los vecinos se encuentran ya colocados, por lo que se elige el primer nodo de la partícula S_1 que no haya sido colocado en el vector.

S_1	0	3	5	9	6	4	1	2	8	7
S_2	7	9	6	4	8	2	1	3	5	0

Quedando:

S_p		5	7	9	6	4	8	2	1	3

Manipulación ocho: Insertar la ciudad 0 en el casillero faltante.

S_p	0	5	7	9	6	4	8	2	1	3

Finalmente traspasar los datos de la partícula S_p a la partícula S_1.

7.7 Utilización del PSO con SA

Combinando la habilidad de una rápida búsqueda de un óptimo de la metaheurística PSO con la probabilidad de salto de propiedad de SA, [10] diseñó un nuevo algoritmo de marco para resolver el problema de TSP. La idea principal es que primero busque cada partícula la mejor ruta local $p_{in}(k)$ usando el algoritmo SA actualizando la mejor ruta personal individual $p_{in}(k)$ y la mejor ruta global $g_n(k)$, $c_1R_1(k)\cdot\left(p_{in}(k)-x_{in}(k)\right)+c_2R_2(k)\cdot\left(g_n(k)-x_{in}(k)\right)$ en la ecuación (5.1) puede ser vista como la operación de cruce del algoritmo genético (GA) para generar nueva ruta.

Paso 1. Inicialización

Para configurar el tamaño del enjambre "m" e inicializar todas las partículas $C_0, C_1, C_2, ..., C_m$. C_i pueden ser inicializado como: $C_i = \{s(1), s(2), ..., s(n)\}$, cuando $s(i) = j$ significa que la ciudad j es visitada en orden i y $S(n+1) = s(1)$.

Configurar T_0, T_θ, α (de la ecuación 6.7) y L. T_0 es la temperatura inicial. Cuanto más alta sea la temperatura inicial, el resultado es el mejor. T_0 es el valor de temperatura más baja. A baja temperatura, cada partícula encuentra la mejor ruta local $p_{in}(k)$. α es el coeficiente de enfriamiento que es una constante azarosa entre 0 y 1. L es el número máximo de iteraciones en una cierta temperatura.

Paso 2. Búsqueda de nueva ruta

Cada partícula C_i busca para sí la mejor ruta local $p_{in}(k)$. En una iteración, cada partícula C_i genera una nueva ruta C_i en su área local y a continuación, de acuerdo con la norma de aceptación de SA, decide si acepta la nueva ruta o no. Después de L iteraciones, cada partícula tiene su mejor ruta local $p_{in}(k)$. Aquí, el intercambio de método se utiliza para generar una nueva ruta.

A continuación se describe el procedimiento:

N ciudades están numeradas como 0, 1, 2, ..., $(n-1)$, respectivamente. Si, $s[u]$ es el número de la ciudad correspondiente a la que visita el vendedor, en el ú-esimo orden de la visita de secuencia, será el del número de orden. Dos ciudades $s[u]$, $s[v]$ son seleccionadas aleatoriamente en la ruta C_i y las ciudades que están en la posición u y v de la partícula se intercambian, mientras que el orden de visitar las otras ciudades sigue siendo el mismo. Tras el intercambio, una nueva ruta C_i es generada.

Si la secuencia de orden de visita a las ciudades sigue la ruta C_i es:

$$...s[u-1]s[u]s[u+1]...s[v-1]s[v]s[v+1]...$$

Después de intercambiar el orden de visita de las ciudades "u" y "v" la nueva ruta $C_{i'}$ es:

$$...s[u-1]s[v]s[u+1]...s[v-1]s[u]s[v+1]...$$

Paso 3. Actualización de mejor ruta

Actualizar la mejor ruta personal $p_{in}(k)$ y la mejor ruta global $g_n(k)$.

Comparar el valor fitness de $p_{in}(k)$ y $g_n(k)$. La ruta con menor valor entre C_i y $C_{i'}$ será la nueva ruta C_i de la partícula. Si esta nueva ruta es menor que la menor ruta encontrada hasta entonces reemplazará a C_{ipbest}.

$$g_n(k) = \begin{cases} p_{in}(k) & f(p_{in}(k)) < f\left(g_n(k)\right) \\ g_n(k) & f(p_{in}(k)) > f\left(g_n(k)\right) \end{cases}$$

Después de la actualización de cada partícula del mejor valor personal, podemos obtener el mejor valor global C_{gbest}.

Paso 4. Obtener nueva ruta

Obtenga la nueva ruta C_{new} para el cruce de la operación.

La partícula C_i cruza con la nueva $p_{in}(k)$ y $g_n(k)$ separadamente que se actualicen. Hay cuatro estrategias de cruce revisados en la literatura y uno de ellos es adoptado en el trabajo. Habiendo dos rutas, $old_1 = 123456789$ y $old_2 = 987654321$. En primer lugar, seleccionar al azar un cruce en la zona old_2, a continuación, inserte el número de ciudad en el cruce a la zona old_1 en la misma posición que en old_2, finalmente eliminar la duplicación de las ciudades en old_1 que aparecen fuera de la zona de cruce. Por ejemplo, el cruce en la zona old_2 elegido es 7 6 5 4, después de la operación de cruce de la nueva ruta será 1 2 7 6 5 4 3 8 9.

Paso 5. Cálculo de nueva temperatura

Calcular la nueva temperatura T especificada en la ecuación (24). Si $T \le T_\theta$ el algoritmo encuentra el mejor camino $g_n(k)$ llega a la final. De lo contrario, repetir el paso 2.

7.8 Criterio de detención

El criterio de iteración solo se produce después de un número determinado de iteraciones, para este caso se probará, en un comienzo, con valores de iteración de $\{50, 75, 100\}$ y, para comparar los resultados con el paper, los valores de iteración serán de $\{30000\}$ para 101 ciudades y de $\{40000\}$ para 200 ciudades. También se puede detener el algoritmo al alcanzar un valor mínimo del fitness, siendo este valor, un valor particular a la función del PTSP a buscar.

7.9 Tamaño de la población

Se evaluará el algoritmo en un comienzo, según los tamaños de la población de $\{10, 20, 40\}$; valores que fueron tomados de referencia para observar que cantidad de partículas que es más conveniente a utilizar.

7.10 Vecindad

Como PSO puede optar por vecindad local o global. Si se opta por vecindad global, es decir, toda partícula es vecina del resto de partículas que conforman el enjambre. Esta elección se sustenta en dos razones:

- Para converger rápidamente hacia las soluciones del problema.
- Al tener una vecindad global, el algoritmo mantiene una forma general.

CAPÍTULO 8

COMPARACIÓN ENTRE EL USO DE LOS ALGORITMOS DE PSO V/S PSO CON SA

En este capítulo se hace una comparación entre los algoritmos PSO y el algoritmo hibrido de PSO con SA para ver si este último mejora la calidad de los resultados finales para poder ocupar éste como base para el proyecto de tesis.

Para este fin se utilizarán los archivos bays29 y eil101 con 29 y 101 ciudades respectivamente. Estos archivos fueron sacados de la página www.tsp.gatech.edu para comparar los resultados obtenidos por estos algoritmos y ver cuál de estos se acerca más al resultado óptimo encontrado por ellos.

Este modelo de PSO con SA es apuntado a la deficiencia del algoritmo básico del PSO, es decir, la fácil captura de mínimos locales que hace que no pueda desplazarse hacia otros mínimos, estancando los resultados obtenidos por éste sin mejorar los resultados alcanzados.

Aquí se utiliza la idea del modelo [10] que propone un avanzado algoritmo PSO con SA para aplicar este algoritmo para resolver el problema del TSP. El núcleo del algoritmo es basado sobre el algoritmo PSO y su variación con SA apunta a la obtención de la generación de la partícula y su mejora en el desplazamiento.

También; el método de SA es usado para frenar la degeneración del enjambre PSO e incrementar la diversidad de las partículas.

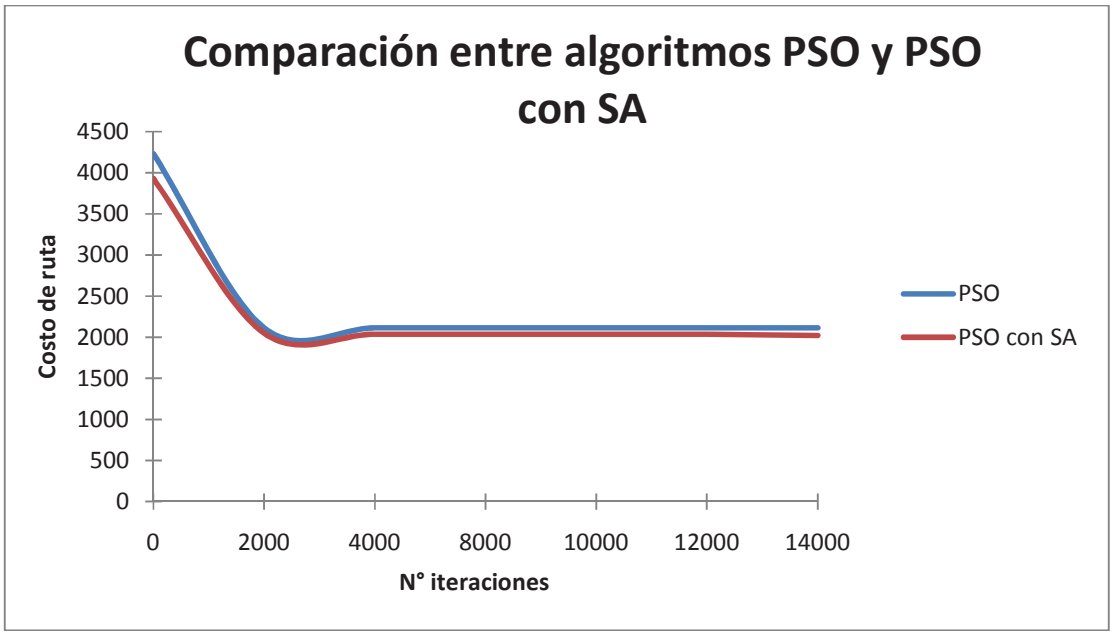


Figura 8.1. Gráfico de comparación entre implementación con PSO y PSO con SA usando archivo bays29.

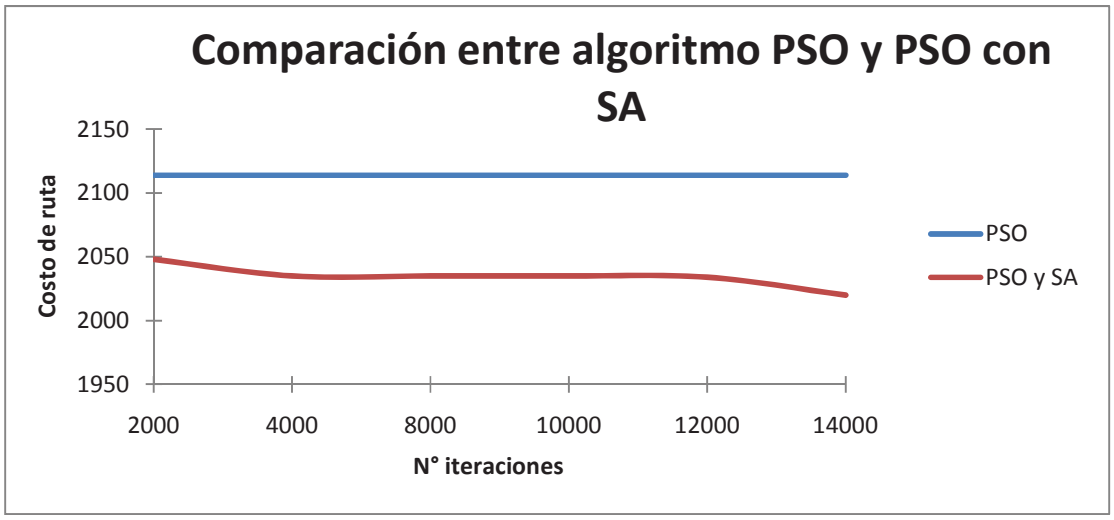


Figura 8.2. Otro Gráfico de comparación entre implementación con PSO y PSO con SA usando archivo bays29.

Las figuras anteriores muestran la comparación de ambos algoritmos. La primera figura muestra la evolución de las partículas usadas y la obtención de los mejores resultados obtenidos (mostrados en el eje Y) versus la cantidad de iteraciones que se han utilizado (eje X).

En este último gráfico se puede apreciar más claramente que el PSO básico provoca estancamiento en los resultados sin producir cambios que puedan llegar al óptimo deseado; en cambio con la combinación de las heurísticas de PSO con SA provoca que el algoritmo

no se estanque en mínimos locales ampliando la búsqueda de resultados, mejorándolos y llegando realmente al óptimo deseando en la página de TSP.

Para la comparación de la mejora del resultado entre el PSO y PSO con SA se utilizó también el archivo eil101, con 101 nodos, que es usado en [5] para el logro de resultados del Problema del Vendedor Viajero Probabilístico por este autor.

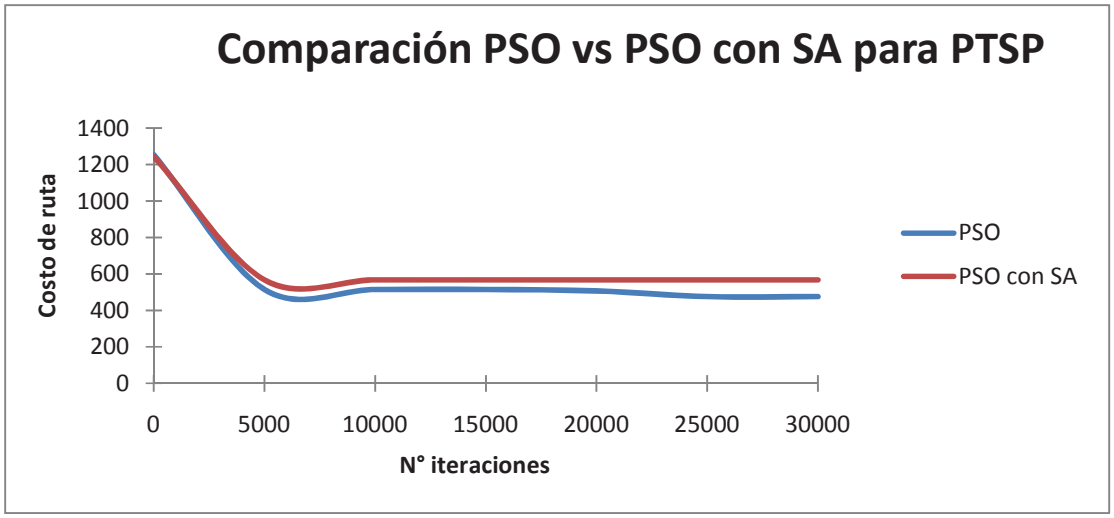


Figura 8.3. Gráfico de comparación entre implementación con PSO vs PSO con SA usando archivo eil101.

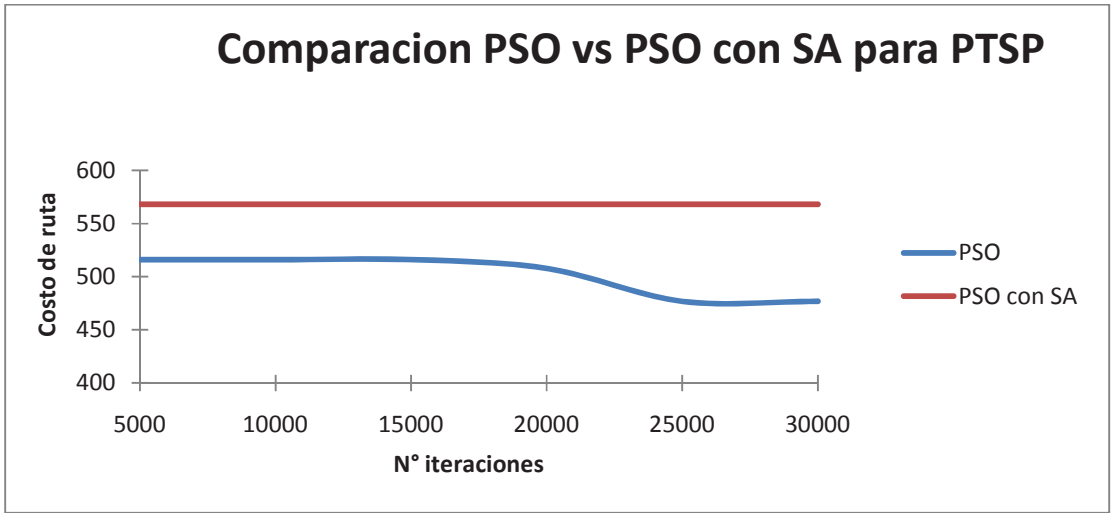


Figura 8.4. Acercamiento a gráfico de comparación entre implementación con PSO vs. PSO con SA usando archivo eil101.

Como se puede observar al igual que en el caso anterior, el algoritmo PSO básico permanece estancado en un mínimo local lo que le impide encontrar mejores rutas; en cambio PSO con SA produce que éste no se estanque en mínimos locales, encontrando

mejores rutas; por lo que pueden ser usadas estas dos heurísticas (PSO y SA) hibridizadas para hallar una posible solución al problema del Vendedor Viajero Probabilístico.
Se pueden observar tres aspectos importantes en los gráficos mostrados en este capítulo:

1. La curva con que el PSO con SA se aproxima al óptimo vs. el estancamiento que presenta PSO solo, sin producir cambios a lo largo del tiempo.

2. La mejora que produce desde un comienzo, el algoritmo en competencia, al ser hibridizado.

3. La lentitud con que el algoritmo de PSO con SA funciona para mejorar los resultados alcanzados vs. los resultados logrados usando solo PSO; es que, durante las primeras 10.000 iteraciones en PSO con SA su temperatura es alta haciendo que los cambios en la partícula también sean altos pero, no así su mejora de los resultados ya que la memoria del enjambre prevalece no encontrando mejoras sustanciales de nivel global, teniendo los mejores resultados a partir de la iteración 10.000 en adelante ya que la memoria de la partícula personal como la del enjambre, tiene más información adquirida durante este tiempo haciendo que el salto provocado por SA sea mejor porque la temperatura desciende más lentamente provocando saltos más selectivos.

También hay que tomar en cuenta que a menor cantidad de población el cambio es menos notorio, ya que es menos la cantidad de iteraciones usadas para llegar al mejor resultado.

Con esta experiencia se puede demostrar la conveniencia de la incorporación de SA al modelo de la solución, lo que mejora los resultados obtenidos en comparación al modelo de PSO solo.

También se toma como base PSO con SA ya que obtuvo un óptimo al cabo de 10 veces que se corrió el programa; en cambio al usar PSO no se pudo llegar, ni se logró aproximar al óptimo inclusive si éste se aumentaba en la cantidad de iteraciones y la cantidad de partículas para el mismo caso.

CAPÍTULO 9

IMPLEMENTACIÓN DEL TSP EN PSO CON SA

Para probar la capacidad de la metaheurística de enjambre de partículas se utilizó la distancia de diferentes ciudades para calcular el costo mínimo que debe recorrer éste para abarcar todas las ciudades que tiene que visitar.

9.1 Implementación de TSP con PSO y SA

Para llevar a cabo la implementación se utilizaron las ecuaciones dadas en el capítulo 2 para su implementación y además del diseño de la solución propuesta en el capítulo 7, utilizando 4 archivos con costes de ciudades. Estas se comparan, el fitness, consigo mismo y se tomaron los mejores resultados luego de las 10 veces que se inicio el programa. Se tomaron para las iteraciones la cantidad de 10, 20 y 40 partículas y, el número de repeticiones usadas son 50, 75 y 100. Los números colocados en rojo son aquéllos que en pocas iteraciones pudieron llegar al resultado óptimo propuesto por la página.

TABLA 9.1 RESULTADOS OBTENIDOS DEL TSP CON PSO Y SA COMPARADOS CON EL ÓPTIMO DE LA PÁGINA DE TSP.

N°	archivo	Optimo	50 iteraciones			75 iteraciones			100 iteraciones		
			10 part	20 part	40 part	10 part	20 part	40 part	10 part	20 part	40 part
1	fri26	937	954	957	955	957	957	955	966	984	937
2			971	937	937	968	964	964	940	966	970
3			981	944	955	1088	944	960	982	946	953
4			946	981	960	993	1061	960	937	946	955
5			1090	991	965	949	959	990	984	960	953
6			1068	982	955	1037	966	937	957	976	937
7			1023	1010	978	950	959	953	990	941	955
8			964	957	975	1008	937	955	976	955	955
9			989	958	937	960	980	955	953	973	955
10	fri26	937	972	998	961	986	971	955	999	952	955
1	bays29	2020	3522	3286	3364	2894	3059	3022	3531	3270	2906
2			3682	3153	3268	3695	3494	3003	3643	3392	2686
3			3482	3146	3488	2982	2988	3249	3055	3112	2753
4			3451	3004	2987	3321	3040	3245	3172	2035	2699
5			3549	3268	3374	3134	2968	3177	3161	3221	3015
6			3477	3153	2818	3098	3323	2961	3344	3059	3015
7			3493	3276	3031	3033	3143	3048	2988	3118	3007
8			3680	3081	3101	3377	2874	3122	3009	3289	3115
9			3518	3289	3185	3331	3133	3096	3124	3289	2973
10	bays29	2020	3695	3192	3150	3109	3098	3166	3347	3208	3113

TABLA 9.2. RESULTADOS OBTENIDOS: TSP CON PSO Y SA COMPARADOS CON EL ÓPTIMO DE LA PÁGINA DE TSP.

N°	archivo	Optimo	50 iteraciones			75 iteraciones			100 iteraciones		
			10 part	**20 part**	**40 part**	**10 part**	**20 part**	**40 part**	**10 part**	**20 part**	**40 part**
1	**Swiss42**	**1273**	1366	1357	1369	1430	1419	1329	1436	1368	1321
2			1425	1443	1310	1409	1330	1384	1524	1321	1299
3			1425	1511	1332	1517	1371	1333	1446	1427	1321
4			1462	1346	1415	1385	1373	1367	1373	1324	1335
5			1490	1481	1420	1529	1373	1355	1532	1321	1306
6			1436	1481	1351	1460	1446	1355	1390	1321	1313
7			1533	1386	1313	1428	1403	1321	1530	1338	1386
8			1533	1474	1336	1428	1448	1341	1477	1459	1317
9			1518	1416	1430	1473	1396	1346	1451	1392	1313
10	**Swiss42**	**1273**	1616	1377	1348	1501	1437	1421	1419	1357	1343
1	**eil101**	**629**	827,64	806,98	795,68	770,19	805,2	778,63	796,65	781,46	723,75
2			819,52	819,69	823,8	869,52	796,23	774,52	818,73	760,32	735,58
3			882,19	826,31	799,89	863,67	751,66	765,06	806,54	789,25	735,06
4			833,81	785,18	818,95	844,72	822,35	767,75	831,74	806,81	747,3
5			873,18	811,57	792,82	854,63	785,15	752,79	815,5	788,17	718,08
6			856,88	830,95	779,51	796,67	882,35	797,73	772,36	762,86	718,08
7			863,68	792	739,11	843,38	778,95	768,64	815,01	815,4	752,38
8			875,23	802,63	819,16	803,6	841,81	767,79	846,72	806,63	769,54
9			844,86	826,87	781,95	864,51	800,44	742,15	836,41	796,93	760,51
10	**eil101**	**629**	838,69	783,53	809,72	861,43	819,1	756,7	781,52	782,12	755,85

Aunque se probaron con 50, 75 y 100 repeticiones los mejores resultados fueron obtenidos al aumentar el número de repeticiones, como se muestra en la tabla, ya que tienen mayores posibilidades de encontrar un fitness mejor.

También se puede apreciar que a menor cantidad de ciudades a buscar no es necesario una gran cantidad de iteraciones ni una gran cantidad de partículas para la búsqueda del óptimo puesto que ésta se logra fácilmente.

9.2 Mejora en el algoritmo de PSO y SA

Para mejorar la distancia entre los vecinos, y así mejorar el resultado final obtenido por el algoritmos, es que se ha agregado a éste un ordenamiento entre la elección de sus vecinos propuesto en el capítulo 7.7. Así se logra mejorar, al aumentar la cantidad de iteraciones, el resultado final obtenido.

La siguiente tabla muestra los resultados obtenidos al usar los mismos archivos anteriormente con el fin de comparar el resultado alcanzado por éstos.

TABLA 9.3. RESULTADOS OBTENIDOS: TSP CON PSO Y SA MEJORADO, COMPARADO CON EL ÓPTIMO DE LA PÁGINA DE TSP.

N°	archivo	Optimo	50 iteraciones			75 iteraciones			100 iteraciones		
			10 part	20 part	40 part	10 part	20 part	40 part	10 part	20 part	40 part
1	fri26	937	973	937	937	971	955	955	957	959	948
2			937	955	955	937	937	963	974	948	937
3			963	960	959	955	963	955	955	957	960
4			965	982	955	982	937	955	967	964	955
5			960	989	970	963	977	937	957	955	955
6			964	964	971	960	951	937	971	953	937
7			937	960	955	955	981	955	963	937	951
8			986	971	937	955	955	955	970	964	955
9			1002	948	970	970	957	937	977	937	937
10	fri26	937	975	955	960	966	973	937	937	971	960

TABLA 9.4. RESULTADOS OBTENIDOS: TSP CON PSO Y SA MEJORADO, COMPARADO CON EL ÓPTIMO DE LA PÁGINA DE TSP.

N°	Archivo	Optimo	50 iteraciones			75 iteraciones			100 iteraciones		
			10 part	20 part	40 part	10 part	20 part	40 part	10 part	20 part	40 part
1	bays29	2020	2219	2195	2100	2053	2159	2082	2146	2045	2085
2			2098	2113	2170	2211	2178	2174	2127	2140	2132
3			2215	2141	2142	2174	2132	2091	2174	2049	2051
4			2215	2167	2193	2226	2117	2134	2103	2152	2129
5			2178	2136	2115	2127	2117	2160	2103	2159	2066
6			2178	2078	2083	2066	2043	2063	2151	2114	2123
7			2223	2164	2103	2289	2099	2104	2111	2139	2085
8			2185	2108	2204	2176	2099	2090	2159	2096	2030
9			2091	2139	2110	2096	2218	2105	2122	2079	2135
10	bays29	2020	2215	2134	2089	2174	2183	2170	2167	2074	2109
1	Swiss42	1273	1473	1537	1404	1461	1356	1390	1386	1404	1409
2			1468	1408	1379	1402	1422	1372	1413	1409	1367
3			1401	1461	1468	1365	1436	1438	1413	1423	1401
4			1503	1378	1439	1437	1342	1422	1331	1324	1377
5			1444	1460	1363	1437	1427	1376	1447	1394	1349
6			1503	1460	1430	1485	1512	1338	1445	1404	1353
7			1463	1458	1364	1494	1392	1422	1445	1421	1350
8			1516	1492	1421	1494	1454	1440	1438	1409	1394
9			1467	1492	1454	1473	1448	1406	1529	1419	1363
10	Swiss42	1273	1459	1430	1370	1545	1493	1392	1393	1413	1335
1	eil101	629	850,21	828,61	835,64	843,71	807,82	817,75	797,63	790,91	782,84
2			831,58	864,57	802,41	807,38	832,84	809,94	799,09	788,63	771,5
3			831,11	840,54	857,84	837,23	827,6	820,05	770,81	793,6	804,64
4			383,58	807,16	804,37	834,37	813,31	810,57	840,75	781,79	786,68
5			827,38	807,43	733,08	831,04	786,94	794,13	799,13	796,55	797,37
6			827,38	838,02	819,37	834,37	827,6	813,37	787,17	783,6	787,98
7			808,4	807,43	835,91	818,71	829,84	805,19	857,47	796,55	786,53
8			870,53	827,86	842,41	820,25	832,45	807,53	816,6	764,53	775,05
9			857,24	824,39	837,68	837,23	832,45	805,19	789,47	805,29	789,3
10	eil101	629	857,24	809	801,49	831,04	816,14	803,9	796,29	788,12	790,14

A continuación se muestran los promedios de los resultados obtenidos entre el TSP con PSO y SA vs TSP con PSO y SA mejorado.

TABLA 9.5. MEDIA OBTENIDA DE LOS RESULTADOS LOGRADOS DE LA TABLA 9.2.

fri26	937,00	995,80	971,50	957,80	989,60	969,80	958,40	968,40	956,50	952,50
Bays29	2020,00	3554,90	3184,80	3176,60	3197,40	3112,00	3108,90	3237,40	3099,30	2928,20
Swiss42	1273,00	1480,40	1427,20	1362,40	1456,00	1399,60	1355,20	1457,80	1362,80	1325,40
eil101	629,00	851,57	808,57	796,06	837,23	808,32	767,18	812,12	789,00	741,61

TABLA 9.6 MEDIA OBTENIDA DE LOS RESULTADOS OBTENIDOS DE LA TABLA 9.3.

fri26	966,2	962,1	956,9	961,4	958,6	948,6	962,8	954,5	949,5	966,2
Bays29	2181,7	2137,5	2130,9	2159,2	2134,5	2117,3	2136,3	2104,7	2094,5	2181,7
Swiss42	1469,7	1457,6	1409,2	1459,3	1428,2	1399,6	1424	1402	1369,8	1469,7
eil101	794,465	825,501	817,02	829,533	820,699	808,762	805,441	788,957	787,203	794,465

TABLA 9.7. PORCENTAJE DE MEJORA OBTENIDA ENTRE TSP CON PSO Y SA MEJORADO VS TSP CON PSO Y SA.

fri26	**1,13%**
bays29	**32,75%**
Swiss42	**-1,59%**
eil101	**-1,04%**

La tabla anterior muestra en que porcentaje mejoró ó empeoró el algoritmo de "TSP con PSO y SA mejorado" con respecto al tradicional. En este caso, al cabo de 100 iteraciones, se puede ver me mejora su búsqueda para pocas ciudades; en cambio, para una gran cantidad de ciudades este disminuye. Esto es, porque las partículas siguen inestables en solo 100 iteraciones versus la gran cantidad de posibles combinaciones que tiene esta para hacer el recorrido.

Se puede sacar como conclusión favorable que al tener pocas ciudades que recorrer no se necesita un gran número de iteraciones, ya que el algoritmo mejorado encuentra rápidamente el fitness propuesto alcanzado una mejora, por ende en el tiempo necesario que tiene que pasar el programa en resolver el TSP.

En cambio, se puede observar que hay un leve alejamiento en el resultado utilizando "PSO con SA mejorado" cuando son pocas las iteraciones aplicadas al algoritmo, ya que, éste mejora los resultados de forma más estable y pareja que la versión anterior a ésta, por lo que es necesario aumentar la cantidad de iteraciones para llegar a un resultado mejor. La proporción de iteraciones propuesta está dada más abajo.

9.3 Proporción en el número de iteraciones

La proporción para la obtención del número de iteraciones, mostrada en la figura 9.1 se sacó en forma proporcional respecto a la cantidad de ciudades participantes, las que con 100 ciudades da 30.000 iteraciones; para 200 ciudades, 40.000 iteraciones. Las iteraciones

serán proporcionales al número de ciudades, excepto para el archivo fri26 el que con 100 iteraciones logra, igualar el óptimo dado por la página de TSP.

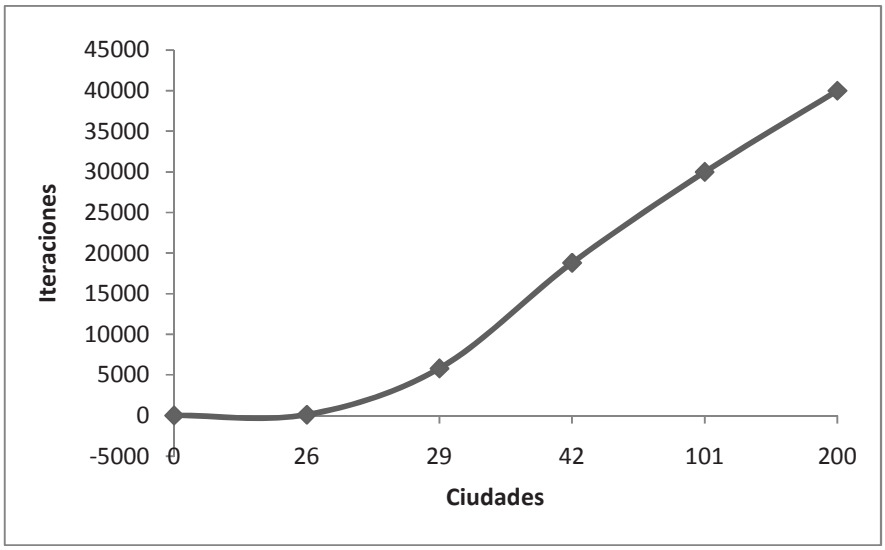


Figura 9.1. Proporción de número de iteraciones con respecto a la cantidad de ciudades.

La siguiente tabla (9.8) muestra los mejores resultados obtenidos por el algoritmo de PSO con SA. Se mantiene como constante el número de partículas. También muestra la tabla el porcentaje de error del algoritmo con respecto a la mejor ruta y por ende mejor fitness encontrado por la página de TSP.

TABLA 9.8. MEJORES RESULTADOS OBTENIDOS CON METAHEURÍSTICAS PSO Y SA VS. MEJORES RESULTADOS (FITNESS) OBTENIDOS POR LA PAGINA WWW.TSP.GATECH.EDU

Archivo	N° Iteraciones	N° Partículas	Fitness	Costo de ruta	% error
Fri26	100	40	937	937	0
Bays29	8400	40	2020	2020	0
Swiss42	12.600	40	1273	1293	1,57
Eil101	30.000	40	629	729	15,90
CroA200	40.000	40	29368	33002	12,37

En la tabla se puede apreciar que al aumentar el número sobre las 100 ciudades el Fitness obtenido se aleja del propuesto por la página, no acercándose al óptimo dado que se espera.

Se puede observar que a menor número de nodos utilizados para optimizar la ruta, no influye en gran medida el número de partículas y el número de iteraciones que se utilicen, como en el caso del archivo fri26 en cual posee la distancia de 26 ciudades.

Si en cambio, a mayor número de ciudades se necesita mayor cantidad de iteraciones para que los resultados obtenidos puedan llegar a un logro que sea aceptable y que se acerque al

óptimo propuesto por la página mencionada anteriormente, teniendo que ocupar gran cantidad de recursos del computador como tiempo necesario para poder realizar ese gran número de iteraciones.

Con el número de iteraciones mostrado en la tabla 9.8, para los diferentes archivos a tratar, es que se pone a prueba el PTSP para la obtención de los resultados que serán comparados con los resultados del Paper mencionado en el capítulo 7.

CAPÍTULO 10

IMPLEMENTACIÓN DEL PTSP EN PSO CON SA

Se implementó el PTSP con datos sacados de la página TSP [www.tsp.gatech.edu] y que también fueron ocupados por el Benchmarck con el cual se comparan los resultados obtenidos.

Se aplicó el diseño de la solución propuesta en el capítulo 7. Ésta se comparó, con el fitness, consigo mismo y se tomaron los mejores resultados luego de las 10 veces que se inició el programa. Se tomaron para las iteraciones la cantidad de $\{10, 20, 40\}$ partículas y, el número de repeticiones tomadas son $\{50, 75, 100\}$.

Al igual que en el caso del TSP, se puede observar en la tabla que se muestra a continuación, que a mayor número de iteraciones y a mayor número de partículas, su fitness mejora, pero éste, no se acerca al resultado obtenido por otros algoritmos; esto es así, ya que al realizar una baja cantidad de iteraciones versus la gran cantidad de ciudades a recorrer y, la gran cantidad de posibilidades que existe para recorrer estas ciudades es que los resultados obtenidos no son cercanos a los datos propuestos. Hay que tomar en cuenta que al aumentar el número de iteraciones va a llegar a un punto donde no va a mejorar significativamente los resultados haciendo tender a cero la variación en la mejora de estos.

El esquema del PSO que se utiliza para la implementación del PTSP es la topología de vecindad global con actualizaciones síncronas.

TABLA 10.1 RESULTADOS OBTENIDOS: PTSP CON PSO Y SA MEJORADO.

N°	Archivo	Prob	50 iteraciones			75 iteraciones			100 iteraciones		
			10 part	20 part	40 part	10 part	20 part	40 part	10 part	20 part	40 part
1	eil101	0,5	654,29	624,98	618,66	609,76	622,13	623,05	624,13	616,26	626,27
2			617,98	613,61	625,3	633,95	620,05	605,02	608,29	622,9	615,64
3			608,42	592,73	,614,71	627,8	628,99	627,32	589,05	619,4	621,17
4			618,88	624,31	622,97	663,81	600,93	621,68	628,96	624,35	610,56
5			627,28	602,23	622,81	627,12	627,08	627,77	627,98	620,53	611,99
6			622,54	619,19	607,56	624,79	623,28	613,65	626,52	596,93	599,6
7			624,25	625,79	614,71	598,59	623,72	603,23	615,38	618,9	604,22
8			614,91	627,6	613,65	607,16	596,11	595,29	617,91	627,21	625,98
9			624,25	618,5	613,41	611,92	616,47	617,95	626,74	627,83	597,89
10	eil101	0,5	626,67	620,78	625,37	617,05	622,54	610,28	625,58	628,51	618,26
1	eil101	0,75	695,98	738,59	703,08	701,34	708,79	670,21	686,22	688,99	674,89
2			706,64	689,25	678,81	696,1	657,04	709,2	723,04	703,55	696,52
3			750,17	700,29	689,85	722	677,71	710,02	739,55	686,97	699,21
4			716,57	695,51	720,83	711,55	693,79	685,25	712,76	696,82	672,56
5			728,97	715,8	693,2	733,55	691,38	698,8	696,76	718,01	684,35
6			735,3	717,22	723,55	695,95	731,91	674,44	692,01	680,62	687,34
7			724,61	732,24	698,17	701,12	723,77	710,23	736,74	674,34	683,63
8			727,21	679,29	724,43	741,21	691,5	677,55	708,15	689,38	681,08
9			740,19	701,42	720,14	747,96	711,43	692,16	683,53	730,47	695,99
10	eil101	0,75	746,81	683,71	692,16	710,57	732,55	712,3	700,85	670,21	685,35

En ambas tablas muestra la mejoría que resulta de aumentar las iteraciones y la cantidad de partículas.

TABLA 10.1 RESULTADOS OBTENIDOS: PTSP CON PSO Y SA MEJORADO.

N°	Archivo	Prob	50 iteraciones			75 iteraciones			100 iteraciones		
			10 part	20 part	40 part	10 part	20 part	40 part	10 part	20 part	40 part
1	eil101	0,5	654,29	624,98	618,66	609,76	622,13	623,05	624,13	616,26	626,27
2			617,98	613,61	625,3	633,95	620,05	605,02	608,29	622,9	615,64
3			608,42	592,73	,614,71	627,8	628,99	627,32	589,05	619,4	621,17
4			618,88	624,31	622,97	663,81	600,93	621,68	628,96	624,35	610,56
5			627,28	602,23	622,81	627,12	627,08	627,77	627,98	620,53	611,99
6			622,54	619,19	607,56	624,79	623,28	613,65	626,52	596,93	599,6
7			624,25	625,79	614,71	598,59	623,72	603,23	615,38	618,9	604,22
8			614,91	627,6	613,65	607,16	596,11	595,29	617,91	627,21	625,98
9			624,25	618,5	613,41	611,92	616,47	617,95	626,74	627,83	597,89
10	eil101	0,5	626,67	620,78	625,37	617,05	622,54	610,28	625,58	628,51	618,26
1	eil101	0,75	695,98	738,59	703,08	701,34	708,79	670,21	686,22	688,99	674,89
2			706,64	689,25	678,81	696,1	657,04	709,2	723,04	703,55	696,52
3			750,17	700,29	689,85	722	677,71	710,02	739,55	686,97	699,21
4			716,57	695,51	720,83	711,55	693,79	685,25	712,76	696,82	672,56
5			728,97	715,8	693,2	733,55	691,38	698,8	696,76	718,01	684,35
6			735,3	717,22	723,55	695,95	731,91	674,44	692,01	680,62	687,34
7			724,61	732,24	698,17	701,12	723,77	710,23	736,74	674,34	683,63
8			727,21	679,29	724,43	741,21	691,5	677,55	708,15	689,38	681,08
9			740,19	701,42	720,14	747,96	711,43	692,16	683,53	730,47	695,99
10	eil101	0,75	746,81	683,71	692,16	710,57	732,55	712,3	700,85	670,21	685,35

TABLA 10.2 RESULTADOS OBTENIDOS: PTSP CON PSO Y SA MEJORADO.

N°	Archivo	Prob	50 iteraciones			75 iteraciones			100 iteraciones		
			10 part	20 part	40 part	10 part	20 part	40 part	10 part	20 part	40 part
1	KroA200	0,5	38423,38	35273,41	37701,76	37023,2	36671,7	32647,56	34292,75	33937,45	35203,4
2			38895,21	38591,9	34948,9	34096,23	34394,67	36216,76	35455,18	34808,45	30051,25
3			39038,28	38238,39	38213,83	37044,09	35131,52	35255,8	36583,78	36392,95	34223,27
4			37965,58	37745,01	34683,7	39691,52	33179,43	34928,64	35252,37	34671,33	29877,67
5			38798,5	35106,17	37201,11	35613,26	35369,37	36165,04	34993,87	32595,66	35769,15
6			39128,43	36118,88	34117,44	33995,17	34100,85	34887,12	34373,39	35859,47	34344,88
7			37309,15	37165,98	35962,71	37402,07	36443,03	35608,91	35657,57	36047,14	34233,9
8			39839,66	37215,69	35752,29	36310,36	35545,13	32357,83	33039,84	34828,33	30801,59
9			38402,99	38571,04	32098,23	34714,47	33818,7	34883,53	34168,75	33443,62	35111,34
10	KroA200	0,5	34909,08	36569,82	37129,45	38089,6	31797,5	36116,38	34716,97	32651,77	32490,84

Para comparar realmente los resultados obtenidos en el proyecto de título es que se procede a igualar la cantidad de iteraciones tomadas por la competencia. Con esto se pretende encontrar buenos resultados que sirvan para la resolución del problema del vendedor viajero.

Estas repeticiones son utilizadas para poder comparar los resultados obtenidos en este trabajo con los logrados por la competencia. Así es como que para el archivo eil101, con 101 nodos, se toman repeticiones de 30.000 y; para el archivo kroA200, con 200 nodos, se toman repeticiones de 40.000.

Para ambos casos la cantidad de partículas utilizadas es de 40. Esta cantidad de partículas está dentro de la configuración típica del PSO.

10.1 Comparación de resultados Obtenidos vs [5]

Aquí se comparan los resultados obtenidos en el trabajo con los Benchmarck

TABLA 10.3 RESULTADOS OBTENIDOS VS BENCHMARCK.

	PSO con SA	TSP-ACO	Angle-ACO	HS-1Shift	Depth-ACO
eil101	**328,41**	325,903	325,071	**321,732**	322,23
P=0,25					
eil101	**470,21**	467,441	463,57	463,36	**460,563**
P=0,5					
eil101	**570,93**	567,076	564,71	572,11	**564,036**
GP=0,75					
KroA200	**18094,18**	17816,2	17719,7	18517	**17574,6**
P=0,25					
KroA200	**23877,46**	23461,2	23341,5	23999,8	**23327,8**
P=0,5					
KroA200	**27163,87**	27205,6	27179,9	30908,1	**27126,1**
P=0,75					

Los resultados que se muestran en la tabla anterior de PSO con SA son los mejores obtenidos luego de 20 veces que se inició el programa.

Como se puede aprecias en la tabla 10.2, aunque los resultados se acercan bastante a los mejores obtenidos por éste y por las distintas heurísticas ocupadas por el autor, no logran mejorar los resultados encontrados por éste.

TABLA 10.4 PORCENTAJE DE ERROR VS. MEJOR RESULTADO ENCONTRADO POR LOS DISTINTOS ALGORITMOS COMPARADOS.

	PSO con SA
eil101	2,076%
P=0,25	
eil101	2,095%
P=0,5	
eil101	1,222%
GP=0,75	
KroA200	2,956%
P=0,25	
KroA200	2,356%
P=0,5	
KroA200	0,139%
P=0,75	

Se puede observar que, al comparar el mejor resultado obtenido por los demás algoritmos versus el mejor resultado obtenido por el algoritmo hibrido de PSO con SA es que el

porcentaje que se aleja, en promedio, es de aproximadamente de un 1,8% lo que indica que este algoritmo no superan los resultados obtenidos por la competencia, pero sí, se acerca a éstos. Si se aumentara el número de partículas para resolver el problema, los resultados no variarían en gran medida, aumentando sí, el tiempo de ejecución del programa por lo que se opta por tener una cantidad de partículas estándar para el cálculo de resultados de los archivos.

CAPÍTULO 11

CONCLUSIÓN

A lo largo de este documento se presentó la investigación, desarrollo e implementación de una metaheurística distinta a las propuestas para la optimización del Problema del Vendedor Viajero Probabilístico. (PTSP).

Se propone una heurística de baja complejidad, que no afecte significativamente la calidad de las soluciones obtenidas, respecto de aquellas entregadas por heurísticas existentes en la literatura. Bajo esta premisa se analiza el algoritmo de la metaheurística de enjambre de partículas (PSO) hibridizado con Simulated Annealing (SA) que mejora la calidad del resultado final en comparación al usado con la heurística del PSO por sí sola.

PSO no se rige por la ley de Darwin, "la sobrevivencia del más fuerte", sino que se basa en la convivencia y el traspaso de comunicación entre ellas, lo que permite a una partícula obtener información relevante. Además surge como motivación el análisis de la utilización de nuevas estructuras de costos de ruteo.

Para la aplicación de algoritmo PSO con SA se realizaron 20 pruebas con 100 y 200 ciudades utilizando 30000 y 40000 iteraciones respectivamente, para la obtención de la solución final; con una población de 40 partículas, probabilidad homogénea para cada uno de 0,25; 0,5; 0,75, además, se destaca que a medida que la cantidad de individuos en la población aumenta, no varía en gran medida la solución final del resultado obtenido por esta; pero en la medida que aumentaba la cantidad de iteraciones mejoraba el resultado final, acercándose más al resultado comparativo obtenido por el Benchmarck.

Una forma de mejorar el resultado obtenido por esta hibridación de metaheurísticas es el análisis de la inserción de los vecinos asignados en las iteraciones, sacando los mejores resultados de esto para así asignarlos a la partícula.

Se destaca el hecho que el algoritmo de PSO es de rápida convergencia produciendo estancamiento en los resultados obtenidos, por lo que es necesario un algoritmo como Simulated Annealing que hace que los resultados obtenidos sean "dispersados" para que las soluciones obtenidas puedan abarcar mayores resultado sin producir el típico estancamiento del PSO estándar.

Al comparar los resultados obtenidos por este modelo y los alcanzados por su símil en el Benchmarck utilizado, se puede llegar a la conclusión que en algunos casos se acerca a la obtención de los mejores resultados obtenidos por estos últimos haciéndolo un buen estimador.

CAPÍTULO 11. CONCLUSIÓN

En otros trabajos se ha demostrado que una mejora en la función de evaluación del PTSP en comparación con la función de evaluación estándar posee resultados idénticos disminuyendo significativamente los costos de evaluación.

Del trabajo realizado se puede concluir que en el caso de PSO mejora la velocidad de convergencia de las partículas a un resultado final y; en el caso de SA, hace que la función de modificación dé mejores resultados e impide el estancamiento de resultados en óptimos locales que se hacen que se aleje del resultado final deseado.

El estancamiento del PSO se debe principalmente a una disminución de la velocidad de las partículas, al tener un gran número de iteraciones, en el espacio de búsqueda que lleva a un estancamiento del fitness del enjambre.

Tras el trabajo realizado se puede concluir que los objetivos específicos planteados en el comienzo de este documento se pueden dar por cumplidos. Estos objetivos fueron los siguientes:

- Tener una visión acabada del estado del arte del PTSP y sus aplicaciones para ser representado el problema, de manera que se acerque a los resultados obtenidos por las técnicas estudiadas en los datos obtenidos por el Benchmarck.

- Comprender el funcionamiento de la metaheurística de Enjambres de Partículas (PSO) y Simulated Annealing (SA).

- Desarrollar un modelo que genere una ruta a seguir, minimizando distancias y/o costos del problema; y.

- Evaluar y contrarrestar el rendimiento de la metaheurística propuesta con metaheurísticas vistas en el trabajo.

Una vez dado por cumplidos los objetivos específicos se concluye que el objetivo general planteado también al comienzo del documento, se puede dar por cumplido. Este objetivo general consistía en lo siguiente:

Resolver Problema del Vendedor Viajero Probabilístico (PTSP) a través de la metaheurística de enjambre de partículas (PSO) y Simulated Annealing (SA).

Como trabajo a futuro se proponer usar otra metaheurística para hibridizar con el PSO de modo de comparar y mejorar los resultados obtenidos a lo largo de este trabajo.

CAPÍTULO 12

BIBLIOGRAFÍA

[1] Applegate, Bixby, D., Chvátal, R., Cook, V. (2006) The Traveling Salesman Problem.

[2] Bertsimas, D. (1988). Probabilistic combinatorial optimization problems, Ph.D. dissertation, Massachusetts Institute of Technology, MA, USA.

[3] Bianchi, L.(2006). Ant Colony Optimization and Local Search for the Probabilistic Traveling Salesman Problem: A case study in stochastic combinatorial optimization. Universite Libre de Bruxelles, Brussels, Belgium.

[4] Bianchi, L., Dorigo, M.,Gambardella, L. M., Gutjahr, W. J. (2008) A survey on metaheuristics for stochastic combinatorial optimization. Universite Libre de Bruxelles, Brussels, Belgium.

[5] Branke, J. Y Guntsch, M. (2004) Solving the Probabilistic TSP with Ant Colony Optimization. Journal of Mathematical Modelling an Algorithms 3: 403-425.

[6] Clerc, M., Kennedy J., "The particle swarm – Explosion, stability, and convergence in a multidimensional complex space", IEEE Transactions on Evolutionary Computation, Vol. 6, No. 1, February 2002, pp. 58-73.

[7] Durán Ledezma, J (2007) Optimización por enjambre de Partículas para reducción de distorsión no lineal en sistemas OFDM. Escuela de Ingeniería Informática, Pontificia Universidad Católica de Valparaíso, CHILE.

[8] Eiben, A.E, Marchiori, E. (2004)Evolutionary Algorithms with On-the-Fly Population Size Adjustment. Lecture Notes in Computer Science (LNCS), Springer-Verlag.

[9] Eiben, A.E., Michalewicz, Z. (2007) Parameter Control in Evolutionary Algorithms. Studies in Computational Intelligence (SCI), Springer-Verlag.

[10] Fang, L, Chen P. (2007) Particle Swarm Optimization with Simulated Annealing for TSP. Proceedings of the 6th WSAS Int. Conf. On Artifical Intelligence, Knowledge Engineering an Data Bases, Corfu Island, Greece.

[11] Jaillet, P. (1985) The Probabilistic Traveling Salesman Problems. Operations Research Center. PhD Thesis, MIT, Cambridge, MA.

[12] Kennedy,J., Eberhart, R.C. (1995) Particle swarm optimization, Proceedings of the IEEE International Conference on Neural Networks-ICNN'95, Perth (Australia).

[13] Kennedy, J.,Eberhartm, R.C., Y. Sh (2001) Swarm Intelligence. The Morgan Kaufmann Series in Artificial Intelligence, Morgan Kaufmann, San Francisco-California.

[14] Lamas Vilches, A. (2007) Un análisis asintótico para una Aproximación de la mejora Local 2-P-opt del Problema del Vendedor Viajero Probabilístico. Tesis de Magíster en Ciencias de la Ingeniería, Escuela de Ingeniería. Pontificia Universidad Católica de Chile.

[15] Liu, H. (2007) "A fuzzy adaptive turbulent particle swarm Optimization". School of Computer Science and Engineering, Dalian Maritime University, China.

[16] Liu, Yu-Hsin (2008) Solving the Probabilistic Travelling Salesman Problem Based on Generic Algorithm with Queen Selection Sceme. Department of Civil Engineering, National Chi Nan University, Taiwan.

[17] Liu, Yu-Hsin (2005) A hybrid scatter search for the probabilistic traveling salesman problem. Department of Civil Engineering, NationalChi, Nan University, Taiwan.

[18] McLaughlin, Michael P.; "Simulated Annealing;" Dr. Dobb's Journal; September 1989; pp. 26-37. http://home.att.net/~srschmitt/sa_demo/SA-demo_1.html

[19] Millonas M.M. (1994) Swarms, phase transitions, and collective intelligence. Proceedings of Artificial life III, Vol. XVII, SFI Studies in the Sciences of Complexity, Addison-Wesle.

[20] Mora Fontana, Francisco (2004) Algoritmo colaborativo entre Tabu Search y Simulated Annealing para la resolución del Problema del Vendedor Viajero. Escuela de Ingeniería Informática, Pontificia Universidad Católica de Valparaíso, CHILE.

[21] Muñoz, Mario A., López, Jesús A., y Caicedo, Eduardo F. (2008) Inteligencia de enjambres: Sociedades para la solución de problemas (una versión). Revista Ingeniería e Investigación Vol 28, Universidad del Valle, Colombia.

[22] Ozcan, E., Mohan, C. K. (1999). Particle swarm optimization: surfing the waves. Proceedings of the 1999 Congress on Evolutionary Computation-CEC99, Washington (USA).

[23] Pérez López, Jesús R. (2005) Contribución a los métodos de optimización basados en procesos naturales y su aplicación a la medida de antenas en campo próximo. Tesis doctoral. Departamento de Ingeniería de comunicaciones. Universidad de Cantabria.

[24] Wallace, K. (1994) Stochastic programming. Wiley, Chichester, UK, 1994. Wiley has released the copyright on the book, and the authors made the text available to the scientific community: it can be Downloaded fo rfree at http://www.unizh.ch/ior/Pages/Deutsch/Mitglieder/Kall/bib/ka-wal-94.pdf.

[25] Wang, Qiang, Xiong, Lei (2006) Improved Particle Swarm Algorithm for TSP Based on the Information Communication and Dynamyc Work Allocation. Asian Journal of Information Techonology 5.

[26] Y. Shi, R.C. Eberhart, (1998) Parameter selection in particle swarm optimization, Proceedings Evolutionary Programming VII. 7 International Conference, EP98, San Diego, USA.

[27] Kirkpatrick S. (1983) Simulated Annealing. Sci, Vol.220, pp.671-680.

[28] A. Abraham, H. Guo, H. Liu, "Swarm Intelligence: Foundations, Perspectives and Applications", Studies in Computational Intelligence (SCI), Springer-Verlag, vol. 26, pages 3-25, November 2006.

[29] Sánchez, H. (2001) "Algoritmo Simulated Annealing inteligente aplicado a la optimización del imán principal de una máquina de resonancia magnética de imágenes". Centro de Biofísica Médica, Universidad de Oriente, Cuba.

ANEXO 1. Código Fuente

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#define N_dimension 101
#define N_particula 40
#define ITERACIONES 30000
#define fitness_TSP 329
#define probabilidad 0.75

//---------------------------------------------------------------------
int particula[N_particula][N_dimension];
int velocidad[N_particula], vector_gbest[N_dimension];
int best[N_particula];
double valor_dist[N_particula];
int vector[N_dimension], vector_SA[N_dimension];
int pos_part[N_particula], vector_vec[4];

int gbest;
double fitness, T_baja;

double matriz[N_dimension][N_dimension];
int matriz_datos[N_dimension][2];

//---------------------------------------------------------------------
void leer_archivo();
void calcula_matriz();

void crear_particulas();
void calculo_distancia();

void Burbuja();
void Buscar_vecinos(int pos_der, int pos_izq, int k, int i);
int insertar_vecinos(int pos_der, int pos_izq, int suma);

void burbuja_der(int pos_der);
int insertar_vecino(int pos);
void Busca_particula(int pos_der, int i);

void Simulated_Annealing(int i, int N_iter);
void PSO(float aux_dist, int i);

void Insertar_der(int pos_der, int k, int i);
void Insertar_izq(int pos_izq, int k, int i);
void Disminuir_dist(int pos, int pos_ant1, int pos_ant2);
```

```
void Imprimir_mejor_ruta(int N_iter);
void Imprimir_matriz();

//---------------------------------------------------------------------
int main(int argc, char *argv[])
{
    int i, l, j, k, pos_der, pos_izq;
    int N_iter=0, suma=0, valor;
    double aux_dist, distancia, q, cu;

    srand((unsigned)time(NULL));

    leer_archivo(); //lee archivo de datos .tsp
    calcula_matriz(); //saca valor de distancia entre nodos
 //   Imprimir_matriz(); //Imprime el valor del calculo de distancia
entre nodos
    crear_particulas(); //nodos sin repetir
    calculo_distancia(); //saca distancia de las particulas

      //Inicio de construccion de particulas
      do{
         for(i=0; i<N_particula; i++)
         {
                 for(j=0; j<N_dimension; j++) vector[j]=-1; //deja el
vector con valor -1
                 for(l=0; l<4; l++) vector_vec[l]=-2; //limpio
vector_vec para usarlo

                 vector[pos_part[i]]=particula[i][pos_part[i]];
                 suma=1; //suma cuenta la cantidad de nodos insertados
en el vector

                 pos_der=pos_part[i]+1;
                 pos_izq=pos_part[i]-1;

                 if(i==(N_particula-1)) //k es la siguiente particula
a comparar
                       k=0;
                 else
                       k=i+1;

            Buscar_vecinos(pos_der, pos_izq, k, i); //buscar vecinos
en la particulas S1 y S2
            Burbuja(pos_izq); //Orden de vecinos por metodo burbuja

            suma=insertar_vecinos(pos_der, pos_izq, suma); //guardo
los 2 nodos menores en derecha e izquierda respectivamente

                 do{
                       //Inserta a la Derecha
                       pos_der++;
                       for(l=0; l<4; l++) vector_vec[l]=-2; //limpio
vector_vec a usar
               if(pos_der < N_dimension){
```

```
                    Insertar_der(pos_der,k,i);
                    //Disminuir_dist(pos_der,(pos_der-1),(pos_der-2));
                    suma++;
                       }

                       //Inserta a la Izquierda
                       pos_izq--;
                       for(l=0; l<4; l++) vector_vec[l]=-2; //limpio
vector_vec a usar
                       if(pos_izq >= 0){
                    Insertar_izq(pos_izq,k,i);
                    //Disminuir_dist(pos_der,(pos_der-1),(pos_der-2));
                    suma++;
                       }
                 }while(suma<N_dimension);

            //calculo nueva distancia en el vector
                 distancia=0;
            q=1-probabilidad;
            for(j=0; j<(N_dimension-1); j++)
                 {
                 for(l=1; l<(N_dimension-1); l++)
                       {
                    cu=(double)pow(q,l-1);
                    valor=(j+l)%N_dimension;

distancia+=(pow(probabilidad,2))*(matriz[vector[j]][vector[valor]])*cu
;
                 }
            }
            //distancia=(pow(probabilidad,2))*distancia;

                 aux_dist=valor_dist[i];
                 valor_dist[i]=distancia;

            Simulated_Annealing(i,N_iter); //Aplicando Simulated
Annealing

                 //guardo le nuevo vector en la particula
                 for(l=0; l<N_dimension; l++)
particula[i][l]=vector[l];

                 PSO(aux_dist,i); //Aplicando Enjambre de particulas
           }

           //mantengo la particula gbest sin modificar
           for(l=0; l < N_dimension; l++)
particula[gbest][l]=vector_gbest[l];

           if(N_iter==0)  printf("\tfitness: %f\n",fitness);
           N_iter++;
           if(N_iter%500==0)  printf("\tN_iter:%d \tfitness:
%f\n",N_iter,fitness);
      }while((N_iter<ITERACIONES) && (fitness>fitness_TSP));
```

```
     Imprimir_mejor_ruta(N_iter); //Imprime mejor ruta encontrada o
al termino de iteraciones

    system("PAUSE");
    return 0;
}

//-----------------------------------------------------------------------
//funciones para abrir archivo y calculo de distancia a la matris de
distancia
//-----------------------------------------------------------------------
void leer_archivo()
{
    int sumar=0, dato;
    FILE *archivo_TSP;

    archivo_TSP=fopen("eil101.tsp", "r");
    if (archivo_TSP == NULL)
        printf("\n\n\t\t\tA R C H I V O   N O   E X I S T E   O
N O   E S   E N C O N T R A D O\n\n");
    else
    {
        printf("\n\n\t\t\tA R C H I V O   E N C O N T R A D O\n\n");
        while((fscanf(archivo_TSP,"%d %d %d", &dato,
&matriz_datos[sumar][0], &matriz_datos[sumar][1]))!=EOF &&
(sumar<N_dimension))//(!feof(archivo_TSP))&&(suma<valor))
        {
            sumar++;
        }
        printf("\t\t\tDatos Leidos: %d\n\n", sumar);
    }

    fclose(archivo_TSP);
}
```

```
//---------------------------------------------------------------------
void calcula_matriz()
{
    double distancia, delta_x, delta_y;
    int I, J;

    for(I=0; I < N_dimension;I++)
        for(J=0; J < N_dimension; J++)
        {
            if(I==J)
                matriz[I][I]=0;
            else
            {
                delta_x=matriz_datos[I][0]-matriz_datos[J][0];
                delta_x=pow(delta_x,2);

                delta_y=matriz_datos[I][1]-matriz_datos[J][1];
                delta_y=pow(delta_y,2);

                distancia=delta_x+delta_y;
                distancia=pow(distancia,0.5);

                matriz[I][J]=distancia;
            }
        }
}
```

```
//---------------------------------------------------------------------
//funciones de: creacion de particulas y calculo de distancia
//---------------------------------------------------------------------
void crear_particulas()
{
     int i, j, random, conta, band_part, k;
    //nodos sin repetir
      for(i=0; i<N_particula; i++)
      {
          velocidad[i]=rand()%N_dimension;

        for(k=0; k<N_dimension; k++)
           {
                 random=rand()%N_dimension;
                 band_part=0;
                 conta=0;
                 particula[i][k]=random;

                 if(k!=0)
                 {
                       for(j=0; j<k; j++)
                             if(random == particula[i][j])
                                   band_part=1;
                       if(band_part==1)
//revisar
                       {
                             for(j=0;j<=k;j++)

      if((conta==particula[i][j])||(conta==particula[i][0]))
                                   {
                                         conta++;
                                         j=0;
                                   }
                             particula[i][k]=conta;
                       }
                 }
           }
      }
}
```

```
//---------------------------------------------------------------------
void calculo_distancia()
{
     int i, j, l, valor;
     double distancia, q, cu;
    //saca distancia de las particulas
      fitness=0;
      gbest=0;

      for(i=0; i<N_particula; i++)
      {
          distancia=0;
        q=1-probabilidad;
        for(j=0; j<(N_dimension-1); j++)
           {
           for(l=1; l<(N_dimension-1); l++)
                 {
                cu=(double)pow(q,l-1);
                valor=(j+l)%N_dimension;

distancia+=(pow(probabilidad,2))*(matriz[j][valor])*cu;
            }
        }
           valor_dist[i]=distancia;

           //toma mejor posicion y posicion actual al azar
        //siendo al comienzo la misma
           pos_part[i]=best[i]=rand()%N_dimension;

        //toma mejor fitness como el primero
           if(i==0)
           {
                 T_baja=fitness=distancia;
                 for(l=0; l < N_dimension; l++)
                       vector_gbest[l]=particula[i][l];
        }

           if(distancia < fitness)
           {
                 gbest=i;
                 T_baja=fitness=distancia;
                 for(l=0; l < N_dimension; l++)
                       vector_gbest[l]=particula[i][l];
           }
      }
}
```

```
//---------------------------------------------------------------------
//funciones de la construccion de particulas
//---------------------------------------------------------------------
void Buscar_vecinos(int pos_der, int pos_izq, int k, int i)
{
     int j, part_act;
    //buscar vecinos en la particula S1
    //guardo en el vector vector_vec los vecinos de S1
      if(pos_der <= (N_dimension-1))
        vector_vec[0]=particula[i][pos_der];
    if(pos_izq >= 0)
        vector_vec[1]=particula[i][pos_izq];

    //busca ciudad en particula S2
      for(j=0; j <= (N_dimension-1); j++)
            if(particula[k][j] == vector[pos_part[i]])
            {
                  part_act=j;
                  j=N_dimension;
            }

      //guardo en el vector vector_vec los vecinos
      if(part_act < (N_dimension-1))
        vector_vec[2]=particula[k][part_act+1];
    if(part_act > 0)
        vector_vec[3]=particula[k][part_act-1];
}

//---------------------------------------------------------------------
void Burbuja(int pos_izq)
{
     int j, l=0, aux_vec;
     aux_vec=-2;

    for(j=0;j<4;j++)
        for(l=0;l<3;l++)
        {
            if(vector_vec[l+1] == -2)//intercambiar nodos
inmediatamente si nodo derecho es -2
            {
                aux_vec=vector_vec[l+1];
                vector_vec[l+1]=vector_vec[l];
                vector_vec[l]=aux_vec;
            }
            else  //veo que el nodo posicionado no sea -2
                if((matriz[vector[pos_izq+1]][vector_vec[l]] >
matriz[vector[pos_izq+1]][vector_vec[l+1]]) && (vector_vec[l]!=-2))
                {
                    aux_vec=vector_vec[l+1];
                    vector_vec[l+1]=vector_vec[l];
                    vector_vec[l]=aux_vec;
                }
        }
}
```

```
//---------------------------------------------------------------------
int insertar_vecinos(int pos_der, int pos_izq, int suma)
{
     int j;
     //guardo los 2 nodos menores en derecha e izquierda
     for(j=0; j<4; j++)
     {
         if(vector_vec[j] >= 0)    //veo que exista el vecino
         {
             if(pos_der < N_dimension)
             {
                 vector[pos_der]=vector_vec[j];
                 suma++;
                 if((pos_izq>=0) && (vector_vec[j+1] !=
vector[pos_der]))
                 {
                     vector[pos_izq]=vector_vec[j+1];
                     suma++;
                 }
                 if((pos_izq >= 0) && (vector_vec[j+1] ==
vector[pos_der]))
                 {
                     vector[pos_izq]=vector_vec[j+2];
                     suma++;
                 }
             }
             else
             {
                 vector[pos_izq]=vector_vec[j];
                 suma++;
             }
             j=4;
          }
     }
     return(suma);
}
```

```
//---------------------------------------------------------------------
void burbuja_der(int pos_der)
{
     int j, l, aux_vec;
    //ordeno vector_vec en orden creciente de acuerdo a distancia
    //metodo burbuja
    for(j=0; j<4; j++)
        for(l=0;l<3;l++)
        {
            aux_vec=-2;
            if(vector_vec[l+1] == -2)//intercambio nodos
inmediatamente si el de la izq es -2
            {
                aux_vec=vector_vec[l+1];
                vector_vec[l+1]=vector_vec[l];
                vector_vec[l]=aux_vec;
            }
            else  //veo que el nodo posicionado no sea -2
                if((matriz[vector[pos_der-1]][vector_vec[l]] >
matriz[vector[pos_der-1]][vector_vec[l+1]]) && (vector_vec[l]!=-2))
                {
                    aux_vec=vector_vec[l+1];
                    vector_vec[l+1]=vector_vec[l];
                    vector_vec[l]=aux_vec;
                }
         }
}

//---------------------------------------------------------------------
int insertar_vecino(int pos)
{
     int j, l, bandera=1;
    for(j=0; j < 4; j++)
    {
        if(vector_vec[j] >= 0)
        {
            bandera=0;
            for(l=0; l < N_dimension; l++)
            {
                if(vector[l] == vector_vec[j])
                {
                    bandera=1;
                    l=N_dimension;
                }
            }
            if(bandera == 0)//ciudad vecina no encuentrada en vector
            {             //agregar aca la posicion a guardar del nodo
                vector[pos]=vector_vec[j];
                j=4;
            }
        }
    }
    return(bandera);
}
```

```
//---------------------------------------------------------------------
void Busca_particula(int pos, int i) //busca un nodo no insertado
{
    int j, l, band, aux_vec;
    for(l=0; l<N_dimension; l++)
    {
        band=0;
        aux_vec=particula[i][l];

        for(j=0; j<N_dimension; j++)
            if(vector[j] == aux_vec)
            {
                  band=1;
                  j=N_dimension;
            }
        if(band==0)
        {
              vector[pos]=aux_vec;
              l=N_dimension;
         }
    }
}
```

```
//---------------------------------------------------------------------
//aplicando simulated annealing
//---------------------------------------------------------------------
void Simulated_Annealing(int i, int N_iter)
{
     int selec_1, selec_2, aux
     float T, P, delta_f, alfa, T_ini;
     int l, valor, j;
     float random;
     double distancia, q, cu;
    //copio vector en vecto_SA para producir cambios
      for(l=0; l<N_dimension; l++)
            vector_SA[l]=vector[l];

      //selecciono dos nodos de forma aleatoria
    //para producir cambios de posicion en vector_SA
      selec_1=rand()%N_dimension;

    do{
        selec_2=rand()%N_dimension;
      }while(selec_1 == selec_2);

      aux=vector_SA[selec_1];
      vector_SA[selec_1]=vector_SA[selec_2];
      vector_SA[selec_2]=aux;

    //calculo nueva distancia en el vector_SA
      distancia=0;
            q=1-probabilidad;
            for(j=0; j<(N_dimension-1); j++)
                 {
                 for(l=1; l<(N_dimension-1); l++)
                       {
                    cu=(double)pow(q,l-1);
                    valor=(j+l)%N_dimension;

distancia+=(pow(probabilidad,2))*(matriz[vector_SA[j]][vector_SA[valor
]])*cu;
                 }
            }

      //Cuando "delta(f) < 0" significa que la nueva ruta es más corta
que la anterior,
      //La nueva ruta es siempre aceptada
      if(distancia < valor_dist[i])
      {
            valor_dist[i]=distancia;
            for(l=0; l<N_dimension; l++)
                 vector[l]=vector_SA[l];
      }
      //En el caso de  "delta(f) > 0" , la probabilidad aceptable "p"
se calculará de acuerdo con la ecuación
      else
      {
```

```
            random=(float)rand()/RAND_MAX;

        //posible calendario de refrigeración
            T_ini=valor_dist[i];
          //alfa=(float)rand()/RAND_MAX;
          alfa=0.88;
            T=pow(alfa,N_iter)*T_ini + T_baja;

              delta_f=(-1)*(distancia-valor_dist[i]);
              P=exp(delta_f/T);

              if(P<random)
              {
                    for(l=0; l<N_dimension; l++)
                          vector[l]=vector_SA[l];
                    valor_dist[i]=distancia;
              }
        }
}
```

```
//---------------------------------------------------------------------
//Aplicando PSO
//---------------------------------------------------------------------
void PSO(float aux_dist, int i)
{
     float R1, R2, c1, c2, p_inercia, valor, valor2, V_aux;
//variables del PSO, numero aleatorio
     int l;
    // r numero aleatorion entre [0,1]
      R1=(float)rand()/RAND_MAX;
      R2=(float)rand()/RAND_MAX;

      //Peso iniercial w entre 0,75 y 1 para favorecer exploracion
      do{
            p_inercia=(float)rand()/RAND_MAX;
      }while(p_inercia<=0.75);

      //c1 y c2 son constantes de aceleración cognitiva y social
      //para 2<c1,c2<4 para mejorar la exploracion
      do{
            valor=rand()%4;
            valor2=(float)rand()/RAND_MAX;
            c1=valor+valor2;
      }while(c1<2);

      do{
            valor=rand()%4;
            valor2=(float)rand()/RAND_MAX;
            c2=valor+valor2;
      }while(c2<2);

      V_aux = p_inercia*velocidad[i] + c1*R1*(best[i] - pos_part[i]) +
                  c2*R2*(gbest - pos_part[i]);
      velocidad[i]=V_aux;
      velocidad[i]=velocidad[i]%N_dimension;

      pos_part[i]=pos_part[i]+velocidad[i];

      //restriccion de velocidad acotada a la cantidad de nodos
      if(pos_part[i]<0)            pos_part[i]=0;
      if(pos_part[i]>=N_dimension)
pos_part[i]=pos_part[i]%N_dimension;

      if(valor_dist[i]<aux_dist)
      {
            best[i]=pos_part[i];
            valor_dist[i]<aux_dist

            if(valor_dist[i]<fitness)
            {
                  gbest=i;
                  for(l=0; l < N_dimension; l++)
                              vector_gbest[l]=vector[l];
```

```
                T_baja=fitness=valor_dist[i];
            }
        }
}

//---------------------------------------------------------------------
//funcion de insercion derecha e izquierda
//busca menor distancia entre vecinos. Si no encuentra busca el
primero no insertado
//---------------------------------------------------------------------
void Insertar_der(int pos_der, int k, int i)
{
     int j, part_act, part_act1, bandera;
    //busca la ciudad en particula S1
      for(j=0; j < N_dimension; j++)
          if(particula[i][j] == vector[pos_der-1])
          {
                part_act=j;
                j=N_dimension;
          }

      //busca ciudad en particula S2
      for(j=0; j < N_dimension; j++)
          if(particula[k][j] == vector[pos_der-1])
          {
                part_act1=j;
                j=N_dimension;
          }

      //guardo en el vector vector_vec los vecinos de S1
      if(part_act < (N_dimension-1))
        vector_vec[0]=particula[i][part_act+1];
    if(part_act > 0)
        vector_vec[1]=particula[i][part_act-1];

      if(part_act1 < (N_dimension-1))
        vector_vec[2]=particula[k][part_act1+1];
    if(part_act1 > 0)
        vector_vec[3]=particula[k][part_act1-1];

    //ordeno vector_vec en orden creciente de acuerdo a distancia
    burbuja_der(pos_der);

    //inserto nuevo vecino
    bandera=insertar_vecino(pos_der);

      if(bandera == 1)
      {
          Busca_particula(pos_der, i);
        Disminuir_dist(pos_der,(pos_der-1),(pos_der-2));
    }
}
```

```
//-------------------------------------------------------------------
---
void Insertar_izq(int pos_izq, int k, int i)
{
     int j, part_act, part_act1, bandera;
    //busca la ciudad en particula S1
      for(j=0;j<=(N_dimension-1);j++)
            if(particula[i][j] == vector[pos_izq+1])
            {
                  part_act=j;
                  j=N_dimension;
            }
      //guardo en el vector vector_vec los vecinos
      if(part_act < (N_dimension-1))
        vector_vec[0]=particula[i][part_act+1];
    if(part_act > 0)
        vector_vec[1]=particula[i][part_act-1];

      //busca ciudad en particula S2
      for(j=0;j<=(N_dimension-1);j++)
            if(particula[k][j] == vector[pos_izq+1])
            {
                  part_act1=j;
                  j=N_dimension;
            }

      //guardo en el vector vector_vec los vecinos
      if(part_act1 < (N_dimension-1))
        vector_vec[2]=particula[k][part_act1+1];
    if(part_act1 > 0)
        vector_vec[3]=particula[k][part_act1-1];

    //ordeno vector_vec en orden creciente de acuerdo a distancia
    //metodo burbuja
    Burbuja(pos_izq);

    //veo si algun vecino de los encontrados no esta en el vector
    //se ve de forma ascendente respecto a la distancia
    bandera=insertar_vecino(pos_izq);

    //los vecinos ya se encuentran en el vector por lo que elijo el
primer nodo no insertado
    //de la particula i
    if(bandera == 1)
    {
        Busca_particula(pos_izq,i);
       // Disminuir_dist(pos_izq,(pos_izq+1),(pos_izq+2));
    }

}
```

```
//---------------------------------------------------------------------
void Disminuir_dist(int pos, int pos_ant1, int pos_ant2)
{
    //comparar distancias si distancia es menor en el centro cambiar
orden de insercion
    float dist1, dist2, dist3, suma1, suma2;
    int aux;

    if((pos<N_dimension) && (pos_ant2>=0))
    {
        if((vector[pos] >= 0) && (vector[pos_ant1] >= 0) &&
(vector[pos_ant2] >= 0))
        {
            dist1=matriz[vector[pos]][vector[pos_ant1]];
            dist2=matriz[vector[pos_ant1]][vector[pos_ant2]];
            dist3=matriz[vector[pos]][vector[pos_ant2]];

            suma1=dist1+dist2;
            suma2=dist1+dist3;

            if(suma2 < suma1)
            {
                aux=vector[pos];
                vector[pos]=vector[pos_ant1];
                vector[pos_ant1]=aux;
            }
        }
    }
}

//---------------------------------------------------------------------
void Imprimir_mejor_ruta(int N_iter)
{
     int i, j=0;
     printf("\t\tMejor fitness a encontrar: %d\n", fitness_TSP);
     printf("\t\tParticulas utilizadas: %d\n",N_particula);
     printf("\t\tMEJOR RUTA ENCONTRADA EN ITERACION:%d\n\n", N_iter);
     for(i=0; i<N_dimension; i++)
     {
          j++;
          printf("[%d]->\t", vector_gbest[i]);

          if(j==8){
               printf("\n");
               j=0;
          }
     }
          printf("\n\n\t\tfitness: %f\n\n", fitness);
}
```