# Trabajo de Inteligencia Artificial de Inspiración Biológica: Aplicación del algoritmo de Enjambre de Partículas para resolver el Problema de Enrutamiento de Vehículo

Camila Perez Mosquera, Martha María Del Toro Carballo,
Manuel Santiago Férnandez Arias, Eduardo David Nuñez de Villavicencio Sánchez

**Resumen**

Enjambre de partículas (PSO, Particle Swarm Optimization) es una metaheurística de búsqueda local que se basa en el comportamiento de las partículas en la naturaleza. PSO permite optimizar un problema a partir de una población de soluciones candidatas, denotadas como "partículas", moviendo éstas por todo el espacio de búsqueda según reglas matemáticas que tienen en cuenta la posición y la velocidad de las partículas. El movimiento de cada partícula se ve influido por su mejor posición local hallada hasta el momento, así como por las mejores posiciones globales encontradas por otras partículas a medida que recorren el espacio de búsqueda. El fundamento teórico de esto es hacer que la nube de partículas converja rápidamente hacia las mejores soluciones. En este trabajo se diseñan experimentos para ver el comportamiento de PSO a la hora de resolver el Problema de Enrutamiento de Vehículo con Restricción de Capacidad (CVRP, por sus siglas en ingles) y se comparar los resultados con los de la metaheurística de búsqueda local: Búsqueda de Vecindad Variable (VNS, por sus siglas en inglés).

El Problema de Enrutamiento de Vehículos (VRP, por sus siglas en inglés) es un problema de optimización combinatoria que consiste en planificar recorridos que permitan satisfacer las demandas de un conjunto de clientes geográficamente dispersos. Para ello se cuenta con una flota de vehículos que parten desde uno o varios depósitos centrales. El objetivo del VRP es diseñar rutas para cada vehículo que minimicen o maximicen algún objetivo.

El VRP y sus variantes han sido estudiados durante más de 50 años. Evidencia de esto se pueden encontrar en las referencias [1, 2]. En la actualidad, este es un tema activo de investigación como se puede apreciar en [3, 4, 5]. El VRP es un problema NP-duro [2], por lo que los métodos exactos solo son factibles para resolver problemas de pequeña dimensión. Por ese motivo se han desarrollado diferentes heurísticas para solucionarlos, entre las que sobresalen las metaheurísticas de búsqueda local [6, 7].

Los algoritmos de búsqueda local están basados en "criterios de vecindad" que modifican una solución para obtener otras soluciones. Algunos de estos criterios son *"Mover aleatoriamente un cliente de una ruta a otra"*, *"Reubicar un cliente en su ruta"*, o *"Intercambiar dos clientes cualesquiera minimizando el costo del viaje"* [1, 5].

Una inspección de estos criterios de vecindad permite identificar que en todos están presentes los mismos elementos, como: *seleccionar una ruta aleatoria*, *seleccionar un cliente de una ruta*, o *seleccionar una ruta que cumpla determinadas características*.

Uno de los algoritmos de búsqueda local que se han empleado para resolver instancias de problemas de CVRP es Búsqueda de Vecindad Variable (VNS, por sus siglas en inglés) pues utiliza criterio de vecindad para obtener nuevas soluciones en la vecindad de una solución inicial y así encontrar el

óptimo. Los criterio de vecindades se definen mediante un conjunto de estructuras de entorno y se consideran una cantidad finita.

El uso de VNS para dar solución al VRP, se ha hecho muy común en los últimos años [5, 8, 9] y ha reportado muy buenos resultados como los presentados en [5].

Otra de las metaheurísticas de búsqueda local que se han usado ha sido el algoritmo de enjambre de Partículas (PSO, por sus siglas en inglés). Los métodos PSO se atribuyen originalmente a los investigadores Kennedy, Eberhart [10] y Shi [11]. En un principio fueron concebidos para elaborar modelos de conductas sociales como el movimiento descrito por los organismos vivos en una bandada de aves o un banco de peces. Posteriormente el algoritmo se simplificó y se comprobó que era adecuado para problemas de optimización. Un amplio estudio de las aplicaciones de PSO se puede encontrar en [12]

El trabajo esta estructurado de la siguiente forma: en la sección 1 se define el Problema de Enrutamiento de Vehículo, algunas variantes del mismo. En la seccion 2 se explica detalladamente en que consiste la metaherurística del PSO y como se adapta para resolver el problema de Enrutamiento de Vehículo. En la sección 3 se explica la metaheurística de VNS y como se aplica al PSO. En la sección 5 se definen los experimentos realizados.

# 1. Problema de Enrutamiento de Vehículos

El Problema de Enrutamiento de Vehículos está formado por un conjunto de nodos que deben ser visitados (clientes) y un conjunto de entidades que deben visitar esos clientes (vehículos). Una ruta es el recorrido que realiza un vehículo para visitar a un grupo de clientes en un orden determinado y luego volver al depósito de donde salió.

En el VRP se pueden optimizar distintas funciones como el número de clientes atendidos, el consumo de combustible o el costo total de los recorridos.

Atendiendo a las diferentes funciones a optimizar que se pueden tener en el VRP, a las características que pueden tener los clientes y las de los vehículos empleados en los recorridos, se tienen diferentes variantes de este problema:

- **VRP con restricciones de capacidad**: Todos los vehículos de la flota son idénticos y su capacidad es limitada [13].
- **VRP con flota heterogénea**: La flota está compuesta por vehículos con distintas características [14].
- **VRP con ventanas de tiempo**: Los clientes tienen un horario en el que los vehículos deben visitarlos [7].
- **VRP con recogida y entrega**: Los clientes requieren los servicios de recogida y/o entrega de mercancía [1].
- **VRP con trasbordo**: Existe un cliente que puede ser también depósito y a partir de ese cliente se puede crear nuevas rutas [15, 16].

El VRP y sus variantes han sido estudiados durante más de medio siglo. La primera referencia al VRP la hace Dantzig [17], donde define el problema de enrutamiento de vehículos con capacidad limitada (CVRP), en el que cada vehículo tiene una capacidad que no debe exceder y se desea minimizar el costo total de los recorridos. En [17] se describe una aplicación real de la entrega de gasolina a las

estaciones de servicio y se propone una formulación matemática. Este problema surge como una generalización del problema del Viajante (TSP) y es NP-Duro [2], por lo que solo se han usado métodos exactos para instancias pequeñas y para el resto de los problemas se han usado heurísticas y metaheurísticas. Una de las heurísticas utilizadas son Búqueda de Vecindad Variable (VNS) que se define en la sección 3 y Enjambre de Partículas (PSO) que se define a continuación.

# 2. Enjambre de Partículas

Optimización por Enjambre de Partículas (PSO, por sus siglas en inglés de *Particle Swarm Optimization*) hace referencia a una metaheurística que evoca el comportamiento de las partículas en la naturaleza.

En un principio fueron concebidos para elaborar modelos de conductas sociales, como el movimiento descrito por los organismos vivos en una bandada de aves o un banco de peces. Posteriormente el algoritmo se simplificó y se comprobó que era adecuado para problemas de optimización.

PSO permite optimizar un problema a partir de una población de soluciones candidatas, denotadas como *partículas*, moviendo éstas por todo el espacio de búsqueda según reglas matemáticas que tienen en cuenta la posición y la velocidad de las partículas. El movimiento de cada partícula se ve influido por su mejor posición local hallada hasta el momento, así como por las mejores posiciones globales encontradas por otras partículas a medida que recorren el espacio de búsqueda. A medida que se descubren nuevas y mejores posiciones, éstas pasan a orientar los movimientos de las partículas. El proceso se repite con el objetivo, no garantizado, de hallar en algún momento una solución lo suficientemente satisfactoria.

## 2.1. Descripción de las fases del algoritmo

En términos generales, la estructura de un algoritmo PSO para optimizar (maximizar o minimizar) una función con una o múltiples variables sigue los siguientes pasos:

1. Crear un enjambre inicial de n partículas aleatorias. Cada partícula consta de los siguientes elementos: una posición que representa una determinada combinación de valores de las variables, el valor de la función objetivo en la posición donde se encuentra la partícula, una velocidad que indica cómo y hacia dónde se desplaza la partícula, y un registro de la mejor posición en la que ha estado la partícula hasta el momento y la mejor posición global encontrada por el resto.
2. Evaluar cada partícula con la función objetivo.
3. Actualizar la posición y velocidad de cada partícula. Esta es la parte que proporciona al algoritmo la capacidad de optimización.
4. Si no se cumple un criterio de parada, volver al paso 2.

**Crear partícula:** Cada partícula está definida por una posición, velocidad y valor que varían a medida que la partícula se mueve. Además, también almacena la mejor posición en la que ha estado hasta el momento y la posición de la mejor solución encontrada por el resto. Cuando se crea una nueva partícula, únicamente se dispone de información sobre su posición y velocidad, el resto de los valores no se conocen hasta que la partícula es evaluada.

**Evaluar partícula:** Evaluar una partícula consiste en calcular el valor de la función objetivo en la posición que ocupa la partícula es ese momento. Para poder identificar si una nueva posición es mejor que las anteriores, es necesario conocer si se trata de un problema de minimización o maximización.

**Mover partícula:** Mover una partícula implica actualizar su velocidad y posición. Este paso es el más importante ya que otorga al algoritmo la capacidad de optimizar.

La velocidad de cada partícula del enjambre se actualiza empleando la siguiente ecuación:

$v_i(t+1) = \omega * v_i(t) + c_1 r_1[p_i(t) - x_i(t)] + c_2 r_2[g(t) - x_i(t)]$

donde:

- $v_i(t+1)$: velocidad de la partícula i en el momento $t+1$, es decir, la nueva velocidad.
- $v_i(t)$: velocidad de la partícula i en el momento t, es decir, la velocidad actual.
- $\omega$: coeficiente de inercia, reduce o aumenta a la velocidad de la partícula.
- $c_1$: coeficiente cognitivo (constante de atracción al mejor personal $p_i(t)$).
- $r_1$: vector de valores aleatorios entre 0 y 1 de longitud igual a la del vector velocidad.
- $p_i(t)$: mejor posición en la que ha estado la partícula i hasta el momento.
- $x_i(t)$: posición de la partícula i en el momento t.
- $c_2$: coeficiente social (constante de atracción al mejor global $g(t)$).
- $r_2$: vector de valores aleatorios entre 0 y 1 de longitud igual a la del vector velocidad.
- $g(t)$: posición de todo el enjambre en el momento t, el mejor valor global.

Para comprender cómo se relaciona esta ecuación con el movimiento de la partícula, resulta útil diferenciar tres partes:

- $\omega * v_i(t)$ es la componente de inercia, responsable de mantener a la partícula moviéndose en la dirección en la que lo ha estado haciendo hasta el momento. El valor recomendado del coeficiente de inercia $\omega$ suele ser entre 0.8 y 1.2. Si $\omega < 1$, la partícula se va desacelerando a medida que avanzan las iteraciones, esto se traduce en menor exploración, pero una convergencia hacia el óptimo más rápida. Si $\omega > 1$, la partícula se va acelerando, lo que permite explorar más zonas del espacio de la función, pero dificulta la convergencia.
- $c_1 r_1[p_i(t) - x_i(t)]$ es la componente cognitiva, responsable de que la partícula tienda a moverse hacia la posición donde ha obtenido mejores resultados hasta el momento. El coeficiente cognitivo $c_1$ suele estar acotado en el rango $[0, 2]$, siendo 2 el valor recomendado. $r_1$ es un vector de valores aleatorios entre 0 y 1 (un valor por cada dimensión) que aporta cierto comportamiento estocástico al movimiento de las partículas, mejorando así la capacidad de escapar de mínimos locales.
- $c_2 r_2[g(t) - x_i(t)]$ es la componente social, responsable de que la partícula tienda a moverse hacia la mejor posición encontrada por el enjambre hasta el momento. Puede interpretarse como el *conocimiento colectivo*. El valor del coeficiente social $c_2$ suele estar acotado en el rango $[0, 2]$, siendo 2 el valor recomendado. $r_2$ es un vector de valores aleatorios entre 0 y 1 (un valor por cada dimensión) que aporta cierto comportamiento estocástico al movimiento de las partículas, mejorando así la capacidad de escapar de mínimos locales.
- La magnitud relativa entre la componente cognitiva y la componente social permite regular el comportamiento exploratorio del algoritmo. Cuanto mayor es el valor de $c_1$ respecto a $c_2$, mayor independencia de movimiento tiene cada partícula, lo que permite mayor exploración, pero mayor lentitud en la convergencia. Por el contrario, cuanto mayor es el valor de $c_2$ respecto a $c_1$, más

obligadas están las partículas a moverse hacia la mejor región encontrada hasta el momento, lo que reduce la exploración, pero acelera la convergencia.

- En algunas versiones del algoritmo, $r_1$ y $r_2$ son escalares en lugar de vectores. Multiplicar cada componente de la velocidad por un valor aleatorio distinto añade mayores fluctuaciones al movimiento de las partículas, lo que, aun a riesgo de retrasar la convergencia, suele generar mejores resultados.

Una vez calculada la nueva velocidad, se puede actualizar la posición de la partícula con la ecuación:

$x_i(t+1) = x_i(t) + v_i(t+1)$

Uno de los principales problemas del algoritmo PSO es que las partículas suelen adquirir velocidades excesivamente altas, lo que las lleva a salirse de los límites del espacio de búsqueda o a que sean incapaces de converger en la región óptima. Es en este paso del algoritmo donde más investigaciones y adaptaciones se han hecho. Algunas de las soluciones son:

- Limitar la velocidad máxima que puede alcanzar una partícula. Siendo $[x_{min}, x_{max}]$ los límites inferior y superior del espacio de búsqueda de cada variable, la velocidad máxima que puede alcanzar la partícula en esa dirección es $v_{max} = \frac{k(x_{max} - x_{min})}{2}$ , donde k suele ser un valor entre 0.1 y 1.
- Si el valor de alguna de las variables excede los límites impuestos, se sobrescribe con el valor del límite correspondiente y se reinicia su velocidad a cero.
- Reducción lineal del coeficiente de inercia $\omega$. Esta estrategia consiste en ir reduciendo el coeficiente de inercia a medida que avanzan las iteraciones. En las primeras iteraciones, las partículas tienen mucha capacidad de exploración y, a medida que avanza el proceso, va reduciéndose su velocidad favoreciendo la convergencia. Puede conseguirse este efecto con la ecuación: $\omega_t = \frac{(\omega_{max} - \omega_{min})(t_{max} - t)}{t_{max} - \omega_{min}}$

donde:

- $\omega_t$: coeficiente de inercia en la iteración t.
- $\omega_{max}$: coeficiente de inercia máximo. Valor con el que se inicia el algoritmo. Valor recomendado de 0.9.
- $\omega_{min}$: coeficiente de inercia mínimo. Valor que se alcanza en la última iteración. Valor recomendado de 0.4.
- $t_{max}$: número máximo de iteraciones.

**Crear enjambre:** La creación del enjambre consiste en la inicializar n nuevas partículas.

**Evaluar enjambre** Evaluar un enjambre consiste en calcular el valor de la función objetivo en la posición de cada una de las partículas que lo forman.

**Mover enjambre** Mover el enjambre consiste en actualizar la posición de cada una de las partículas que lo forman.

## 2.2. Formalización del algoritmo

Lo descrito anteriormente puede formalizarse del siguiente modo: sea $f$: $\mathbb{R}^n \rightarrow \mathbb{R}$ la función de coste que se desea minimizar. La función $f$ toma como argumento una solución candidata, representada como un vector de números reales, y da como salida un número real que indica el valor de la función objetivo para la solución candidata obtenida. Las mejores posiciones se corresponden con los mejores valores de la función objetivo $f$. El objetivo es hallar una solución a que verifique $f(a) \leq f(b)$ para todo b en el espacio de búsqueda, lo que implicaría que a es el mínimo global. El proceso inverso, útil en problemas de maximización, puede lograrse considerando una función $h = -f$.

Sea S el número de partículas en la nube, cada una de las cuales tiene una posición $x_i \in \mathbb{R}^n$ en el espacio de búsqueda y una velocidad $vi \in \mathbb{R}^n$. Sea pi la mejor posición conocida de una partícula i, y g la mejor posición global conocida. Un algoritmo PSO básico podría describirse como sigue:

- Para cada partícula $i = 1, \ldots, S$:
  1. Inicializar la posición de la partícula mediante un vector aleatorio uniformemente distribuido: $x_i \rightsquigarrow U(x_{min}, x_{max})$, donde $x_{min} y x_{max}$ son respectivamente el límite inferior y el límite superior del espacio de búsqueda.
  2. Inicializar la mejor posición conocida de la partícula a su posición inicial: $p_i \leftarrow x_i$.
  3. Si $f(p_i) < f(g)$ actualizar la mejor posición global conocida: $g_i \leftarrow p_i$.
  4. Inicializar la velocidad de la partícula: $v_i \rightsquigarrow U(-|x_{up} - x_{min}|, |x_{up} - x_{min}|)$.
  5. Mientras no se cumpla el criterio de parada (por ejemplo: límite máximo de iteraciones, encontrada una solución satisfactoria), repetir:
     - Para cada partícula $i = 1, \ldots, S$, Para cada dimensión $d = 1, \ldots, n$:
       *a*) Elegir números aleatorios: $r_1, r_2 \rightsquigarrow U(0, 1)$
       *b*) Actualizar la velocidad de la partícula:
       $v_i = \omega * v_{(i,d)} + c_1 r_1 (p_{(i,d)} - x_{(i,d)}) + c_2 r_2 (g_d - x_{(i,d)})$
       *c*) Actualizar la posición de la partícula:
       $x_i \leftarrow x_i + v_i$
       *d*) Si $f(x_i) < f(p_i)$ entonces actualizar la mejor posición conocida de la partícula:
       $p_i \leftarrow x_i$.
       *e*) Si $f(p_i) < f(g)$ actualizar la mejor posición global: $g \leftarrow p_i$
       *f*) Devolver $g$ como la mejor solución encontrada.

Los parámetros $\omega, c_1$ y $c_2$ son definidos por un especialista y regulan el comportamiento y la eficacia del método PSO.

La PSO básica suele incurrir fácilmente en óptimos locales. Esta convergencia prematura puede evitarse ignorando la mejor posición global g conocida, y atendiendo en su lugar a la mejor posición l conocida del sub-enjambre *circundante* a la partícula en movimiento. Este sub-enjambre puede definirse geométricamente, por ejemplo: las m partículas más cercanas, o bien de forma social, es decir, como un conjunto de partículas relacionadas, con independencia de la distancia que las separa.

## 2.3. Adaptación del PSO para dar solución a nuestra variante del VRP

En nuestro enfoque definimos que cada partícula está compuesta por su velocidad y posición, el registro de la mejor posición en la que ha estado la partícula hasta el momento y la mejor posición

global encontrada por el resto, justo como el PSO. La partícula representa una solución que consiste en la permutación de los clientes que en principio simboliza el orden de los clientes desde 1 hasta n. Esta representación sigue una pauta similar a la presentada en [18]

Se conoce por el VRP la demanda de cada cliente, la distancia de todos los clientes entre ellos y la distancia que existe entre el almacén y cada cliente. Todos los camiones tienen la misma capacidad.

Como la partícula, no es más que una lista de clientes, para obtener las rutas y la evaluación se siguen las siguientes ideas:

- El objetivo consiste en rellenar cada camión según la demanda de cada cliente, por lo que internamente cada camión lo que tendrá es un subconjunto de clientes cuyas demandas son compensadas en el camión correspondiente.
- Cada camión realiza el recorrido completo empezando desde el almacén hasta el primer cliente de su subconjunto, luego se dirige al segundo cliente y así sucesivamente hasta llegar al último cliente de su subconjunto. En este punto se regresa al almacén.
- La función de evaluación para nuestra adaptación consiste en minimizar la suma del coste (distancia) de cada ruta para cada camión.

Normalmente el PSO se emplea para dar solución a problemas continuos y el VRP es discreto por lo que se tomaron concesiones. Para actualizar la posición de la partícula se le añade la velocidad, pero esto puede dar lugar a resultados continuos. Dado que la partícula de nuestro problema representa clientes, no tiene sentido guardar las soluciones no enteras, puesto que no existe representación física para el cliente 1.5 por solo citar un ejemplo. Algunas alternativas para lidiar con este problema pudiera ser el redondeo de los términos, sin embargo, pueden aparecer números repetidos, o sea, clientes repetidos, introduciendo resultados que no están contemplados en el problema. Por tanto, este problema se abordó de la siguiente forma:

- Para calcular la velocidad en dicho problema se utilizó la formula clásica del PSO. En esta solución propuesta, la velocidad se moduló entre 0 y 30 porque la velocidad tiende a dispararse y de no modularse, daría lugar a valores muy grandes, o sea, se acelera mucho lo que hace que converja más rápido a óptimos locales.
- El movimiento de la partícula está influenciado por el objetivo de acercarse a la mejor posición global.

Dicho esto, para calcular el desplazamiento de una partícula se utiliza su posición actual, la velocidad y el objetivo a alcanzar. Se usaron tres enfoques:

1. La partícula se compara con el óptimo global, posición a posición, y en dependencia de la velocidad y una probabilidad aleatoria se cambia el orden del cliente de esa posición. En profundidad, si la velocidad de la partícula es mayor que 20 (número que se seleccionó tras varios experimentos del cuadro 5 de la página 17) se le da la posibilidad de cambiar con una probabilidad entre 0 y 1, si la velocidad es menor a 20 no se modifica. Para el caso que resulte una probabilidad superior a 0.5 cambia en el siguiente sentido: al compararse con el mejor global, aquellas posiciones cuyo valor no coincidieron son sustituidas por el valor que aparece en el mejor global. El valor que antes estaba en la posición que se acaba de sustituir se ubica donde en el vector se encuentra el valor que acaba de reemplazarlo.
2. La mayor diferencia con la primera versión de movimiento que se describió previamente es el hecho de que pasa si la velocidad de una determinada partícula no es mayor que 20. En la versión

1 simplemente no se modificaba nada. En aras de aumentar el espacio de búsqueda, se introdujo un comportamiento alternativo para aquellas partículas cuya velocidad para alcanzar al óptimo sea menor que 20, entonces va a dirigirse a un objetivo, una solución generada completamente aleatoria. El proceso de cambio con este nuevo objetivo se realiza de la misma forma que en la versión 1. Notar que esto no garantiza que se obtengan a priori mejores resultados, sin embargo, si aumenta el espacio de búsqueda.

3. Esta tercera aproximación es una extensión a las dos anteriores, el punto es cuando una partícula ha alcanzado el valor objetivo, no tiene posibilidad de moverse. En este enfoque si una partícula determinada es igual al óptimo, intenta ir a una solución candidata totalmente aleatoria, lo que aumentaría el espacio de búsqueda. El proceso de mutar hacia la nueva solución es el mismo descrito para los enfoques 1 y 2.

Para evitar converger rápidamente a óptimos locales se realizaron perturbaciones sobre el óptimo global para obtener nuevas posibles soluciones. En cada iteración, excepto la primera, se toma el mejor global y se le cambian la cuarta parte de los términos de forma aleatoria. Estos términos se intercambian dos a dos y luego se vuelve a evaluar, si aparece una mejor solución que la existente, se actualiza. Este proceso se realiza 100 veces para perturbar la solución e intentar encontrar un óptimo global que ninguna partícula habría explorado con anterioridad. Notar que en la primera iteración de dicha perturbación se toma como referencia el óptimo global, pero en posteriores iteraciones la referencia será la perturbación obtenida en la iteración anterior.

En el pseudocódigo 1 de la página 8 se presenta un pseudocódigo del algoritmo PSO Mutado usado en los experimentos.

```
particle : Partícula que se va mutar
_posList = take len(particle)
i = 0
while i < len(_postList) do
    swap(particle[i],particle[i + 1])
    i− > i + 2
end
posible_best = global_best
for i = 0 to 100 do
    posible_best = mutate(posible_best)
    _eval = eval(posible_best)
    if _eval ≤ global_best_eval then
        global_best = posible_best
    end
end
```

**Algorithm 1:** Algoritmo PSO Mutado

En la siguiente sección se presentará el algortimo de VNS.

# 3. Búqueda de Vecindad Variable

Búsqueda de Vecindad Variable (VNS) es una metaheurística que se basa en el cambio sistemático de entorno dentro de una búsqueda local [19]. Las primeras referencias que se tienen sobre esta metaheurística son: [20] de 1995 y [21] de 1997 donde usan VNS para dar solución al problema del viajante

considerando la existencia o no de *backhauls*, esto significa, en el caso del VRP, que los vehículos primero entregan las mercancías a los clientes y después recogen lo que los clientes deben devolver. Por tanto, para el Problema del Viajante se debe considerar una demanda para las ciudades.

VNS está basada en dos principios básicos:

1. Un mínimo local con una estructura de entornos no lo es necesariamente con otra.
2. Un mínimo global es un mínimo local con todas las posibles estructuras de entornos.

Los principios anteriores sugieren el empleo de varias estructuras de entornos en las búsquedas locales para abordar un problema de optimización, pues si se encuentra un óptimo con todas las estructuras, entonces se podría afirmar que es el óptimo global del problema (al menos con los criterios de vecindad que se usaron).

Para resolver un problema utilizando VNS es necesario definir qué es una solución, cuándo una solución es mejor que otra, qué criterios de vecindad se deben usar, y qué estrategia seguir para explorar la vecindad de la solución actual.

A continuación se formalizarán los elementos presentados previamente. Para ello, el conjunto de todas las posibles soluciones se denota como $X$, y el conjunto de soluciones vecinas de $x$, para un criterio de vecindad $V$, se denota como $V(x)$.

**Definición 1** Dada una función $f$ que se desea minimizar, una solución $y$, es un mínimo local de $f$ para el criterio de vecindad $V$, si no existe una solución $x \in V(y)$ tal que $f(x) < f(y)$.

**Definición 2** Dada una función $f$ que se desea minimizar, una solución $x^* \in X$ es un mínimo global de $f$, si no existe una solución $x \in X$ tal que $f(x) < f(x^*)$.

En VNS se usan estructuras de entorno para cambiar de soluciones. La siguiente definición formaliza esta idea. **Definición 3** Cada estructura de entorno (o criterio de vecindad) en el espacio de soluciones $X$ es una aplicación $\eta : X \to 2^X$ que a cada solución $x \in X$ le asigna un conjunto de soluciones $\eta(x) \subset X$, que se dicen **vecinas** de $x$.

La Búsqueda de Vecindad Variable parte de una solución inicial y un conjunto de estructuras de entorno. Utilizando uno de los criterios de vecindad se obtiene una solución vecina de la actual. Si la nueva solución es mejor, se cambia y se regresa a usar el primer criterio. Si no, se generan soluciones vecinas usando el resto de los criterios hasta que no sea posible mejorar la solución actual con ninguno de ellos.

Tomando en cuenta el principio: *un mínimo global es un mínimo local para todas las posibles estructuras de entornos*, se puede decir que esa solución es un mínimo global, al menos para el conjunto de entornos utilizados.

La idea de VNS se expresa en forma de pseudocódigo en el algoritmo 2 de la página 10.

En el algoritmo, en la línea 6, se define **Exploración** como explorar el $k$-ésimo entorno de $x$ y obtener $x'$. En dependencia de cómo se explore ese entorno y de como se seleccione la mejor solución se tienen distintas variantes del algoritmo. En la siguiente sección se presentan algunas de las estrategias de exploración de la vecindad y en la sección 4 se definen las estrategias de selección de la mejor solución dentro de la vecindad.

## 3.1. Estrategia de exploración de una vecindad

La forma en que se exploran soluciones en el entorno de la solución actual define una estrategia de exploración.

```
1  η_1, η_2, ..., η_{k_max} : Conjunto finito de estructuras de entorno.
2  x : Solución Inicial
3  while No se cumplan las condiciones de parada do
4  |  k = 1
5  |  while k ≠ k_max do
6  |  |  Exploración: Explorar η_k (k-ésimo entorno de x)
7  |  |  Selección: Seleccionar x' de las soluciones dentro de η_k
8  |  |  if x' es mejor que x then
9  |  |  |  x ← x'
10 |  |  |  k ← 1
11 |  |  end
12 |  |  else
13 |  |  |  k ← k + 1
14 |  |  end
15 |  end
16 end
```

**Algorithm 2:** Algoritmo VNS

En la literatura [1] se reportan las siguientes:

- **Exploración exhaustiva**: Se analizan todos los vecinos partiendo de una solución inicial.
- **Exploración aleatoria**: Se analiza solamente un subconjunto aleatorio de todos los posibles vecinos.

Una vez explorada la vecindad, de acuerdo a alguna estrategia de exploración, hay que seleccionar uno de los vecinos analizados para devolverlo. La forma en que esto se hace depende de la estrategia de selección que se use. Las estrategias de selección permiten escoger los mejores vecinos dentro de una vecindad, proporcionando posibles candidatos a óptimos del problema en cuestión.

En la siguiente sección se hace una definición formal de estas estrategias y se presentan algunas de ellas.

# 4. Estrategia de selección

Al explorar una vecindad, casi siempre se busca quedarse con la mejor solución vecina. Esto puede provocar que el algoritmo converja a un óptimo local. En muchos casos solo aceptar una buena solución, aunque no sea la mejor, puede dar muy buenos resultados [22].

Un paso importante a la hora de explorar una vecindad es saber qué vecino devolver como óptimo de la misma. Este proceso recibe el nombre de estrategia de selección.

En la literatura [1] se pueden encontrar los siguientes ejemplos:

- **Mejor vecino**: Se devuelve el mejor vecino encontrado entre todos los explorados.
- **Mejor vecino aleatorio**: Se devuelve un vecino aleatorio entre todos los vecinos que mejoren la solución actual.
- **Mejor vecino aleatorio N**: Se selecciona un conjunto de $N$ vecinos que mejoren la solución actual y se devuelve uno de manera aleatoria.

- **Primera mejora**: Se devuelve el primer vecino que mejore la solución inicial.

Estas estrategias de selección se usan en el pseudocódigo 2 de la página 10 en la línea 7.

A continuación veremos los experimentos y los resultados obtenidos al resolver el VRP usando PSO y VNS.

# 5. Experimentos

En los experimentos realizados se usaron los problemas clásicos que aparecen en **NEO Networking and Emerging Optimization** [23].

Los problemas de este sitio tiene entre 30 y 100 clientes y se dividen en varios grupos en dependecia de como se calculan las distancias o de cuan dispersos estan los clientes. Por ejemplo el problema: `A-n33-k5` tiene 33 clientes y la solución óptima tiene 5 rutas. En [23] se hace una explicación detallada de que significa cada letra.

Los experimentos se corrieron en una computadora **I5-7200** con 4GB de RAM y un disco SSD de 256GB.

Para realizar los experimentos de PSO se siguen los pasos:

1. Se generan una determinada cantidad de soluciones aleatorias, número que se corresponde con el número de partículas del experimento en cuestión.
2. Para cada solución inicial se ejectua el algoritmo el número de iteraciones definidos para el experimento .
3. De los pasos 1 y 2 se realizan 30 repeticiones , se calcula el valor mínimo, tomando como óptimo el mínimo entre todos los valores obteninos como solucón en cada una de las 30 repeticiones.

Aplicando la adaptación del PSO, descrita en este trabajo, sobre estos problemas se obtuvieron los siguientes resultados.

**Primer enfoque implementado para el movimiento de la partícula** Los resultados de este experimentos se pueden ver en los cuadros 1 y 2 de las páginas 12 y 13

**Segundo enfoque implementado para el movimiento de la partícula** Los resultados de este experimentos se pueden ver en las tablas 3 y 4 de las páginas 14 y 15

**Tercer enfoque implementado para el movimiento de la partícula** Los resultados de este experimentos se pueden ver en las tablas 5 y 6 de las páginas 16 y 17

En las tablas se refleja que a medida que el número de partículas y la cantidad de iteraciones aumenta por lo general, se obtienen mejores resultados, es decir, el valor mínimo y el medio decrecen y por tanto constituyen mejores soluciones. Como consecuencia de este aumento de partículas e iteraciones, el tiempo medio de ejecución se ve afectado ya que al realizar más búsquedas, inevitablemente el tiempo crecerá en aras de explorar para encontrar la mejor solución posible. Dada la limitada capacidad de cómputo de la que se dispone, no se pudieron ejecutar experimentos de mayor envergadura, es decir, con una cantidad de partículas e iteraciones mayor. Sin embargo, a mayor cantidad de partículas, mayor capacidad de exploración y mejores resultados, y si además se aumenta la cantidad de iteraciones, mejor aún en el caso que se disponga de recursos adecuados y tiempo para la ejecución de los experimentos.

Si bien los enfoques abordados para el movimiento de las partículas generan soluciones válidas, se debe destacar que el primer enfoque obtiene generalmente mejores resultados que los otros. No

| Problema | # Iteraciones | # Partículas | Mínimo | Media | Tiempo Medio |
|---|---|---|---|---|---|
| **A-n32-k5.vrp** | 30 | 10 | 1570 | 1731 | 0.09 |
| **A-n32-k5.vrp** | 30 | 100 | 1131 | 1378 | 0.23 |
| **A-n32-k5.vrp** | 100 | 10 | 1509 | 1701 | 0.22 |
| **A-n32-k5.vrp** | 100 | 100 | 1192 | 1387 | 0.75 |
| **A-n32-k5.vrp** | 1000 | 10 | 1456 | 1668 | 3.01 |
| **A-n32-k5.vrp** | 1000 | 100 | 1251 | 1395 | 11.04 |
| **A-n33-k6.vrp** | 30 | 10 | 1283 | 1447 | 0.1 |
| **A-n33-k6.vrp** | 30 | 100 | 1028 | 1178 | 0.22 |
| **A-n33-k6.vrp** | 100 | 10 | 1154 | 1389 | 0.32 |
| **A-n33-k6.vrp** | 100 | 100 | 1109 | 1198 | 0.8 |
| **A-n33-k6.vrp** | 1000 | 10 | 1309 | 1407 | 3.21 |
| **A-n33-k6.vrp** | 1000 | 100 | 961 | 1184 | 10.04 |
| **A-n44-k6.vrp** | 30 | 10 | 1986 | 2086 | 0.11 |
| **A-n44-k6.vrp** | 30 | 100 | 1597 | 1708 | 0.29 |
| **A-n44-k6.vrp** | 100 | 10 | 1815 | 2044 | 0.33 |
| **A-n44-k6.vrp** | 100 | 100 | 1507 | 1693 | 1.09 |
| **A-n44-k6.vrp** | 1000 | 10 | 1721 | 2029 | 3.93 |
| **A-n44-k6.vrp** | 1000 | 100 | 1558 | 1713 | 11 |
| **A-n65-k9.vrp** | 30 | 10 | 3117 | 3367 | 0.15 |
| **A-n65-k9.vrp** | 30 | 100 | 2511 | 2752 | 0.41 |
| **A-n65-k9.vrp** | 100 | 10 | 3031 | 3249 | 0.58 |
| **A-n65-k9.vrp** | 100 | 100 | 2546 | 2805 | 1.57 |
| **A-n65-k9.vrp** | 1000 | 10 | 2990 | 3259 | 5.56 |
| **A-n65-k9.vrp** | 1000 | 100 | 2530 | 2767 | 15.87 |
| **A-n80-k10.vrp** | 30 | 10 | 4245 | 4441 | 0.17 |
| **A-n80-k10.vrp** | 30 | 100 | 3410 | 3795 | 0.48 |
| **A-n80-k10.vrp** | 100 | 10 | 4032 | 4302 | 0.78 |
| **A-n80-k10.vrp** | 100 | 100 | 3523 | 3791 | 1.89 |
| **A-n80-k10.vrp** | 1000 | 10 | 4005 | 4271 | 6.54 |
| **A-n80-k10.vrp** | 1000 | 100 | 3508 | 3822 | 19.12 |

Cuadro 1: Resultados del primer experimento con 10 y 100 partículas

obstante, hay que destacar que el segundo y tercer enfoque tienen una mayor capacidad de búsqueda lo que implica que con más iteraciones, más partículas y mejores enfoques para abordar el problema de la convergencia en óptimos locales, se esperan mejores resultados.

Se puede observar que a medida que se aumentan las iteraciones, con la misma cantidad de partículas, se obtienen en principio, mejores resultados en las tablas anteriores. Como consecuencia, el tiempo de ejecución aumenta debido al creciente número de iteraciones.

Seguidamente, se presentan las siguientes gráficas donde se realizan comparaciones entre las versiones implementadas y otros análisis de interés.

Se puede observar en la figura 1 de la página 18 que a medida que se aumenta la cantidad de partículas, con la misma cantidad de iteraciones, se obtienen en principio, mejores resultados.

En la figura 2 y 3 de la página 19 y 20 se muestra como a medida que aumentan la cantidad de iteraciones se obtienen mejores resultados.

En las figuras 1 , 5 y 6 de las páginas 18 , 21 y 21 se muestran las comparaciones de las diferentes

| Problema | # Iteraciones | # Partículas | Mínimo | Media | Tiempo Medio |
|---|---|---|---|---|---|
| **A-n32-k5.vrp** | 30 | 1000 | 1077 | 1200 | 1.83 |
| **A-n32-k5.vrp** | 30 | 100000 | 849 | 959 | 280.31 |
| **A-n32-k5.vrp** | 100 | 1000 | 1005 | 1173 | 5.93 |
| **A-n32-k5.vrp** | 1000 | 1000 | 1075 | 1209 | 191.45 |
| **A-n33-k6.vrp** | 30 | 1000 | 889 | 1013 | 1.85 |
| **A-n33-k6.vrp** | 30 | 100000 | 778 | 863 | 238.02 |
| **A-n33-k6.vrp** | 100 | 1000 | 961 | 1045 | 6.17 |
| **A-n33-k6.vrp** | 1000 | 1000 | 948 | 1037 | 88.1 |
| **A-n44-k6.vrp** | 30 | 1000 | 1399 | 1485 | 2.37 |
| **A-n44-k6.vrp** | 30 | 100000 | 1139 | 1230 | 267.35 |
| **A-n44-k6.vrp** | 100 | 1000 | 1307 | 1492 | 7.94 |
| **A-n44-k6.vrp** | 1000 | 1000 | 1304 | 1487 | 150.77 |
| **A-n65-k9.vrp** | 30 | 1000 | 2223 | 2425 | 3.7 |
| **A-n65-k9.vrp** | 30 | 100000 | 1852 | 1959 | 279.42 |
| **A-n65-k9.vrp** | 100 | 1000 | 2283 | 2420 | 11.53 |
| **A-n65-k9.vrp** | 1000 | 1000 | 2333 | 2454 | 109.74 |
| **A-n80-k10.vrp** | 30 | 1000 | 3156 | 3376 | 4.22 |
| **A-n80-k10.vrp** | 30 | 100000 | 2666 | 2840 | 344.33 |
| **A-n80-k10.vrp** | 100 | 1000 | 3101 | 3387 | 13.76 |
| **A-n80-k10.vrp** | 1000 | 1000 | 3139 | 3362 | 166.21 |

Cuadro 2: Resultados del primer experimento con 1000 y 100000 partículas

variantes para los problemas de 32, 44 y 80 clientes.

De las tres versiones, la primera es la que mejores resultados obtuvo para el conjunto de experimentos que se realizaron en este trabajo. La segunda y tercera versión exploran más, convergen con más lentitud y en realidad, deberían proporcionar mejores resultados a medida que se aumente la cantidad de iteraciones y partículas, pero se necesitan mejores recursos de cómputo y tiempo para la ejecución de una mayor cantidad de experimentos. Es importante destacar que a medida que se aumenta la cantidad de clientes, los resultados se alejan del óptimo esperado por lo que se explicó anteriormente y debido a que PSO se emplea mayormente para resolver problemas continuos y el que estamos abordando es discreto, es decir, a la hora de realizar los cálculos estamos discretizando el resultado para satisfacer al problema, pero esto trae como consecuencia que se pierda lo que se estaba ganando con la heurística.

Para realizar los experimentos de VNS se siguen los siguientes pasos:

1. Se generan 30 soluciones iniciales para cada problema usado.
2. Para cada solución inicial se corre el algoritmo 30 veces
3. Para cada una de esas 30 repeticiones del algoritmo se calcula el valor mínimo, de forma tal que se tienen 30 valores mínimos para una misma solución (uno por cada iteración) y el mínimo de esos 30 valores se toma como el valor óptimo obtenido para la solución.

Los criterios usados en VNS son:

- Re-insertar un cliente en su ruta.
- Mover un cliente de una ruta a otra.

| Problema | # Iteraciones | # Partículas | Mínimo | Media | Tiempo Medio |
|---|---|---|---|---|---|
| **A-n32-k5.vrp** | 30 | 10 | 1570 | 1731 | 0.04 |
| **A-n32-k5.vrp** | 30 | 100 | 1496 | 1642 | 0.35 |
| **A-n32-k5.vrp** | 100 | 10 | 1619 | 1681 | 0.19 |
| **A-n32-k5.vrp** | 100 | 100 | 1446 | 1577 | 1.22 |
| **A-n32-k5.vrp** | 1000 | 10 | 1477 | 1577 | 2.12 |
| **A-n32-k5.vrp** | 1000 | 100 | 1385 | 1503 | 6.77 |
| **A-n33-k6.vrp** | 30 | 10 | 1283 | 1447 | 0.04 |
| **A-n33-k6.vrp** | 30 | 100 | 1239 | 1360 | 0.38 |
| **A-n33-k6.vrp** | 100 | 10 | 1308 | 1392 | 0.2 |
| **A-n33-k6.vrp** | 100 | 100 | 1277 | 1347 | 1.32 |
| **A-n33-k6.vrp** | 1000 | 10 | 1259 | 1339 | 2.14 |
| **A-n33-k6.vrp** | 1000 | 100 | 1183 | 1272 | 6.75 |
| **A-n44-k6.vrp** | 30 | 10 | 1986 | 2086 | 0.06 |
| **A-n44-k6.vrp** | 30 | 100 | 1898 | 2001 | 0.48 |
| **A-n44-k6.vrp** | 100 | 10 | 1866 | 2026 | 0.25 |
| **A-n44-k6.vrp** | 100 | 100 | 1837 | 1948 | 1.69 |
| **A-n44-k6.vrp** | 1000 | 10 | 1866 | 1942 | 2.81 |
| **A-n44-k6.vrp** | 1000 | 100 | 1759 | 1859 | 11.14 |
| **A-n65-k9.vrp** | 30 | 10 | 3117 | 3367 | 0.09 |
| **A-n65-k9.vrp** | 30 | 100 | 3064 | 3232 | 0.75 |
| **A-n65-k9.vrp** | 100 | 10 | 3096 | 3268 | 0.37 |
| **A-n65-k9.vrp** | 100 | 100 | 3046 | 3178 | 2.56 |
| **A-n65-k9.vrp** | 1000 | 10 | 3005 | 3182 | 7.78 |
| **A-n65-k9.vrp** | 1000 | 100 | 2934 | 3057 | 17.84 |
| **A-n80-k10.vrp** | 30 | 10 | 4245 | 4441 | 0.11 |
| **A-n80-k10.vrp** | 30 | 100 | 4111 | 4291 | 0.88 |
| **A-n80-k10.vrp** | 100 | 10 | 4235 | 4360 | 0.45 |
| **A-n80-k10.vrp** | 100 | 100 | 3969 | 4208 | 2.52 |
| **A-n80-k10.vrp** | 1000 | 10 | 4078 | 4223 | 8.45 |
| **A-n80-k10.vrp** | 1000 | 100 | 3961 | 4112 | 15.83 |

Cuadro 3: Resultados del segundo experimento con 10 y 100 partículas

- Intercambiar dos clientes.
- Intercambiar dos subrutas.
- Re-insertar una subruta en su ruta.
- Mover una subruta de una ruta a otra.
- Intercambiar dos subrutas invertidas.
- Invertir una subruta e insertarla en su ruta.
- Invertir una subruta e insertarla en otra ruta.
- Seleccionar una subruta e intercambiarla por otra invertida.
- Seleccionar una subruta invertirla e intercambiarla por otra sin invertir.

| Problema | # Iteraciones | # Partículas | Mínimo | Media | Tiempo Medio |
|---|---|---|---|---|---|
| **A-n32-k5.vrp** | 30 | 1000 | 1486 | 1551 | 1.47 |
| **A-n32-k5.vrp** | 30 | 100000 | 1229 | 1397 | 192.42 |
| **A-n32-k5.vrp** | 100 | 1000 | 1385 | 1491 | 5.02 |
| **A-n32-k5.vrp** | 1000 | 1000 | 1375 | 1430 | 77.29 |
| **A-n33-k6.vrp** | 30 | 1000 | 1219 | 1306 | 1.52 |
| **A-n33-k6.vrp** | 30 | 100000 | 1147 | 1205 | 194.44 |
| **A-n33-k6.vrp** | 100 | 1000 | 1181 | 1267 | 5.88 |
| **A-n33-k6.vrp** | 1000 | 1000 | 1185 | 1221 | 60.77 |
| **A-n44-k6.vrp** | 30 | 1000 | 1824 | 1907 | 1.98 |
| **A-n44-k6.vrp** | 100 | 1000 | 1805 | 1868 | 11.76 |
| **A-n44-k6.vrp** | 1000 | 1000 | 1696 | 1810 | 77.73 |
| **A-n65-k9.vrp** | 30 | 1000 | 2961 | 3127 | 2.89 |
| **A-n65-k9.vrp** | 100 | 1000 | 2908 | 3072 | 21.35 |
| **A-n65-k9.vrp** | 1000 | 1000 | 2824 | 2984 | 111.87 |
| **A-n80-k10.vrp** | 30 | 1000 | 3944 | 4153 | 3.64 |
| **A-n80-k10.vrp** | 100 | 1000 | 3880 | 4118 | 30.71 |
| **A-n80-k10.vrp** | 1000 | 1000 | 3860 | 4013 | 207.65 |

Cuadro 4: Resultados del segundo experimento con 1000 y 100000 partículas

- Seleccionar una subruta invertirla y dejarla en la misma posición dentro de la ruta.

Para el caso del VNS como estrategia de exploración de la vecindad se usó exploración exahustiva y como estrategia de selección se usó selección aleatoria.

La comparación de ambos algortimos se hizo en cuanto al valor mínimo encontrado.

En la tabla 5 de la página 17 se presentan los resultados obtenidos con cada algoritmo.

# 6. Conclusiones y recomendaciones

En este trabajo se estudió el comportamiento de PSO y VNS para resolver VRP. Con las implementaciones realizadas se obtuvo como resultados que VNS se comporta mejor que PSO.

Como trabajos futuros se propone realizar una estimación de parámetros de $\omega, c_1 y c_2$ para ver cuales valores se obtienen mejores resultados. Investigar nuevas variantes a implementar en la función de movimiento para evitar la convergencia en óptimo locales. Por ejemplo se pueden tener k particulas que sean los centroides de k cluster de forma que las demas particulas deban en cada ietración acercarse a un cluster de manera aleatoria. De esta forma se podría evitar el cae en optimos locales. También se propone explorar otras formas de perturbación de la solución actual, con el objetivo de obtener nuevas posibles soluciones, que serían candidatas a ser la mejor solución.

# 7. Distribución del Trabajo

Martha y Manuel trabajaron en la parte de PSO y Camila y Eduardo en la parte de VNS.

| Problema | # Iteraciones | # Partículas | Mínimo | Media | Tiempo Medio |
|---|---|---|---|---|---|
| **A-n32-k5.vrp** | 30 | 10 | 1570 | 1731 | 0.14 |
| **A-n32-k5.vrp** | 30 | 100 | 1496 | 1642 | 0.86 |
| **A-n32-k5.vrp** | 100 | 10 | 1619 | 1681 | 0.5 |
| **A-n32-k5.vrp** | 100 | 100 | 1446 | 1577 | 2.87 |
| **A-n32-k5.vrp** | 1000 | 10 | 1477 | 1577 | 5.07 |
| **A-n32-k5.vrp** | 1000 | 100 | 1385 | 1503 | 31.07 |
| **A-n33-k6.vrp** | 30 | 10 | 1283 | 1447 | 0.14 |
| **A-n33-k6.vrp** | 30 | 100 | 1239 | 1360 | 0.87 |
| **A-n33-k6.vrp** | 100 | 10 | 1308 | 1392 | 0.51 |
| **A-n33-k6.vrp** | 100 | 100 | 1277 | 1347 | 2.94 |
| **A-n33-k6.vrp** | 1000 | 10 | 1259 | 1339 | 5.36 |
| **A-n33-k6.vrp** | 1000 | 100 | 1183 | 1272 | 31.31 |
| **A-n44-k6.vrp** | 30 | 10 | 1986 | 2086 | 0.14 |
| **A-n44-k6.vrp** | 30 | 100 | 1898 | 2001 | 0.97 |
| **A-n44-k6.vrp** | 100 | 10 | 1866 | 2026 | 0.78 |
| **A-n44-k6.vrp** | 100 | 100 | 1837 | 1948 | 3.33 |
| **A-n44-k6.vrp** | 1000 | 10 | 1866 | 1942 | 6.41 |
| **A-n44-k6.vrp** | 1000 | 100 | 1759 | 1859 | 35.82 |
| **A-n65-k9.vrp** | 30 | 10 | 3117 | 3367 | 0.2 |
| **A-n65-k9.vrp** | 30 | 100 | 3064 | 3232 | 1.18 |
| **A-n65-k9.vrp** | 100 | 10 | 3096 | 3268 | 0.88 |
| **A-n65-k9.vrp** | 100 | 100 | 3046 | 3178 | 4.08 |
| **A-n65-k9.vrp** | 1000 | 10 | 3005 | 3182 | 8.23 |
| **A-n65-k9.vrp** | 1000 | 100 | 2934 | 3057 | 45.42 |
| **A-n80-k10.vrp** | 30 | 10 | 4245 | 4441 | 0.24 |
| **A-n80-k10.vrp** | 30 | 100 | 4111 | 4291 | 1.42 |
| **A-n80-k10.vrp** | 100 | 10 | 4235 | 4360 | 0.86 |
| **A-n80-k10.vrp** | 100 | 100 | 3969 | 4208 | 4.62 |
| **A-n80-k10.vrp** | 1000 | 10 | 4078 | 4223 | 10.34 |
| **A-n80-k10.vrp** | 1000 | 100 | 3961 | 4112 | 55.17 |

Cuadro 5: Resultados del tercer experimento con 10 y 100 partículas

# Referencias


[1] Alina Fernández Arias. El problema de enrutamiento de vehículos con recogida y entrega simultánea considerando una flota heterogénea. Master's thesis, Facultad de Matemática y Computación. Universidad de La Habana, La Habana, Cuba., 7 2010.

[2] Paolo Toth; Daniele Vigo. *The Vehicle Routing Problem (Monographs on Discrete Mathematics and Applications)*. Monographs on Discrete Mathematics and Applications. SIAM, 2001.

[3] A. Serdar Tasan; Mitsuo Gen. A genetic algorithm based approach to vehicle routing problem with simultaneous pick-up and deliveries, 2012.

[4] I Kucukoglu, S Ene, A Aksoy, and N Ozturk. A green capacitated vehicle routing problem with fuel consumption optimization model. *International Journal of Computational Engineering Research*, 3:16–23, 7 2013.

| Problema | # Iteraciones | # Partículas | Mínimo | Media | Tiempo Medio |
|---|---|---|---|---|---|
| **A-n32-k5.vrp** | 30 | 1000 | 1486 | 1551 | 9.14 |
| **A-n32-k5.vrp** | 30 | 100000 | 1229 | 1397 | 262.15 |
| **A-n32-k5.vrp** | 100 | 1000 | 1385 | 1491 | 30.51 |
| **A-n32-k5.vrp** | 1000 | 1000 | 1373 | 1439 | 89.1 |
| **A-n33-k6.vrp** | 30 | 1000 | 1219 | 1306 | 9.02 |
| **A-n33-k6.vrp** | 30 | 100000 | 1147 | 1205 | 279.18 |
| **A-n33-k6.vrp** | 100 | 1000 | 1181 | 1267 | 29.83 |
| **A-n33-k6.vrp** | 1000 | 1000 | 1170 | 1222 | 92.9 |
| **A-n44-k6.vrp** | 30 | 1000 | 1824 | 1907 | 10.13 |
| **A-n44-k6.vrp** | 30 | 100000 | 1657 | 1764 | 354.08 |
| **A-n44-k6.vrp** | 100 | 1000 | 1805 | 1868 | 33.26 |
| **A-n44-k6.vrp** | 1000 | 1000 | 1701 | 1810 | 123.74 |
| **A-n65-k9.vrp** | 30 | 1000 | 2961 | 3127 | 12.39 |
| **A-n65-k9.vrp** | 100 | 1000 | 2908 | 3072 | 40.43 |
| **A-n65-k9.vrp** | 1000 | 1000 | 2853 | 2973 | 177.68 |
| **A-n80-k10.vrp** | 30 | 1000 | 3944 | 4153 | 14.11 |
| **A-n80-k10.vrp** | 100 | 1000 | 3880 | 4118 | 45.1 |
| **A-n80-k10.vrp** | 1000 | 1000 | 3888 | 4008 | 222.39 |

Cuadro 6: Resultados del tercer experimento con 1000 y 100000 partículas

| Probabilidad de Cambio exitoso | | |
|---|---|---|
| Valor Que Posibilita el cambio | | Probabilidad |
| 10 | | 75 % |
| 15 | | 73 % |
| 20 | | 0.60 % |
| 25 | | 0.60 % |

Cuadro 7: Experimento para seleccionar el valor medio de la velocidad


[5] Amous Marwa; Toumi Said; Jarboui Bassem; Eddaly Mansour. A variable neighborhood search algorithm for the capacitated vehicle routing problem. *Electronic Notes in Discrete Mathematics*, 58, 04 2017.

[6] Hu Xiangpei Wang Zheng, Li Ying. A heuristic approach and a tabu search for the heterogeneous multi-type fleet vehicle routing problem with time windows and an incompatible loading constraint. *Computers and Industrial Engineering*, 11 2014.

[7] Van Woensel Tom Ghilas Veaceslav, Demir Emrah. An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows and scheduled lines. *Computers & Operations Research*, 2 2016.


| **Problemas** | **A-n32-k5** | **A-n33-k6** | **A-n44-k7** | **A-n65-k9** | **A-n80-k10** |
|---|---|---|---|---|---|
| **VNS** | 784 | 742 | 1088 | 1177 | 2540 |
| **PSO** | 849 | 778 | 1139 | 1852 | 2666 |
| **Media VNS** | 823 | 766 | 1167 | 1235 | 2784 |
| **Media PSO** | 959 | 863 | 1230 | 1958 | 2840 |
| **Mínimo reportado en [23]** | 784 | 742 | 937 | 1174 | 1764 |

Cuadro 8: Comparaciones de VNS y PSO

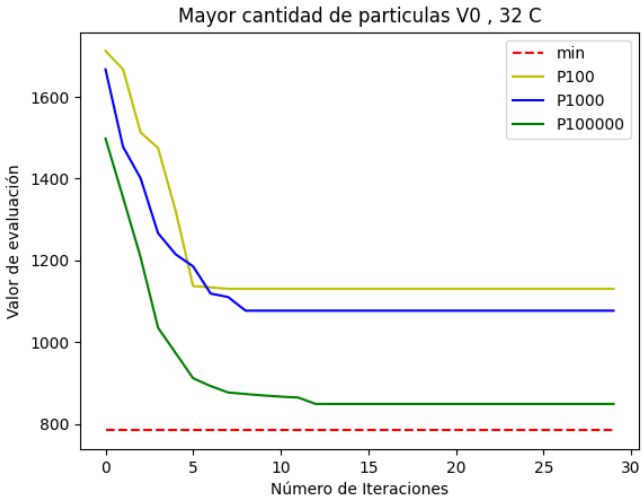


Figura 1: Primer enfoque con una cantidad de partículas igual a 100, 1000 y 100000, 32 clientes y 30 iteraciones


[8] Raca Todosijević, Saïd Hanafi, Dragan Urošević, Bassem Jarboui, and Bernard Gendron. A general variable neighborhood search for the swap-body vehicle routing problem. *Computers & Operations Research*, 78:468–479, 2017.

[9] M Affi, H Derbel, and B Jarboui. Variable neighborhood search algorithm for the green vehicle routing problem. *International Journal of Industrial Engineering Computations*, 9(2):195–204, 2018.

[10] James Kennedy and Russell Eberhart. Particle swarm optimization. In *Proceedings of ICNN'95-international conference on neural networks*, volume 4, pages 1942–1948. IEEE, 1995.

[11] Yuhui Shi and Russell Eberhart. A modified particle swarm optimizer. In *1998 IEEE international conference on evolutionary computation proceedings. IEEE world congress on computational intelligence (Cat. No. 98TH8360)*, pages 69–73. IEEE, 1998.

[12] Riccardo Poli. An analysis of publications on particle swarm optimization applications. *Essex, UK: Department of Computer Science, University of Essex*, 2007.

[13] J. Berger and M. Barkaoui. A new hybrid genetic algorithm for the capacitated vehicle routing problem. *Journal of the Operational Research Society*, 54(12):1254–1262, 2003.

[14] Ochi Luiz Satoru Penna Puca Huachi Vaz, Subramanian Anand. An iterated local search heuristic for the heterogeneous fleet vehicle routing problem. *Journal of Heuristics*, 19, 04 2013.

[15] Benedikt Vornhusen, Xin Wang, and Herbert Kopfer. Vehicle routing under consideration of transhipment in horizontal coalitions of freight carriers. *Procedia CIRP*, 19:117–122, 2014.

[16] YOSHITAKA NAKAO and HIROSHI NAGAMOCHI. Worst case analysis for a pickup and delivery problem with single transfer (numerical optimization methods, theory and applications). 2008.

[17] J. H. Dantzig, G. B.; Ramser. The truck dispatching problem. *Management Science*, 6, 10 1959.

[18] Alejandro Peña. Enrutamiento de vehículos con entrega y recogida simultánea con ventanas de tiempo a través de optimización por enjambre de partículas. 2014.

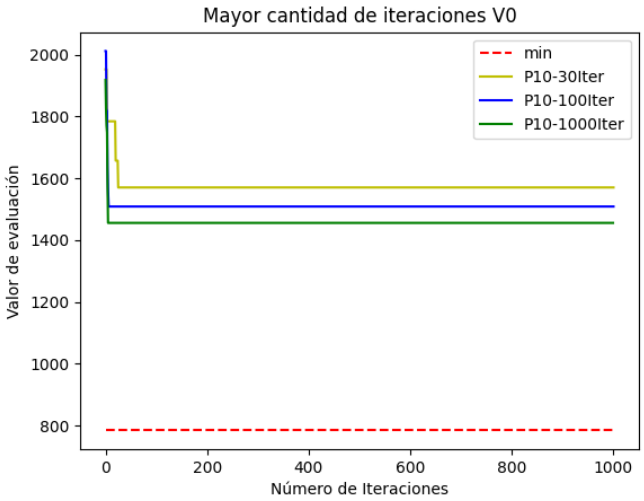


Figura 2: Comparación al aumentar el número de iteraciones para 32 clientes

[19] Pierre Hansen and Nenad Mladenović. First vs. best improvement: An empirical study. *Discrete Applied Mathematics*, 154(5):802 – 817, 2006. {IV} ALIO/EURO Workshop on Applied Combinatorial OptimizationIV ALIO/EURO Workshop on Applied Combinatorial Optimization.

[20] N Mladenovic. A variable neighborhood algorithm–a new metaheuristic for optimization combinatorial. In *Abstract of papers presented at Optimization Days, Montreal*, volume 12, 1995.

[21] Nenad Mladenović and Pierre Hansen. Variable neighborhood search. *Computers & operations research*, 24(11):1097–1100, 1997.

[22] Alina Fernández Arias. Modelo y métodos para el problema de enrutamiento de vehículo con recogida y entrega simultánea, 4 2017.

[23] NEO Networking and Emerging Optimization. Instancias de pruebas para vrp. http://neo.lcc.uma.es/vrp/vrpinstances/capacitated-vrp-instances/, 1 2013.

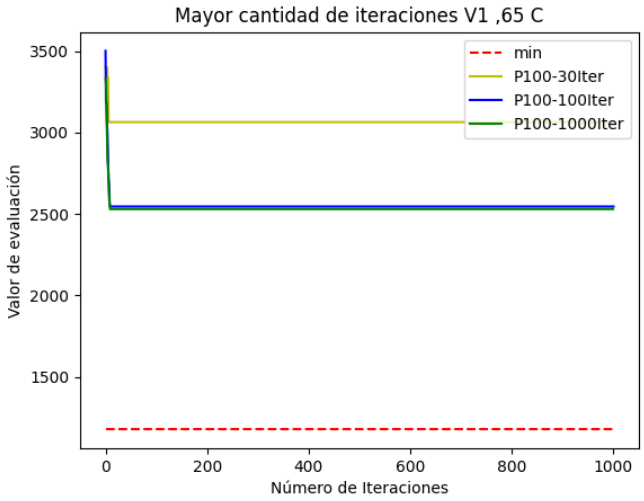


Figura 3: Comparación al aumentar el número de iteraciones para 65 clientes

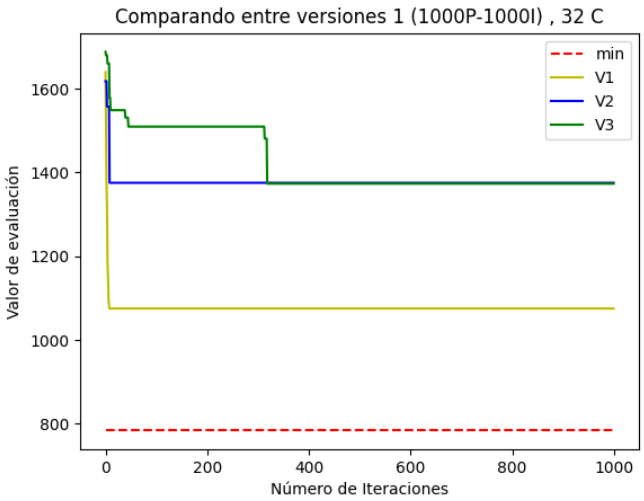


Figura 4: Comparación entre versiones para 1000 partículas, 1000 iteraciones y 32 clientes

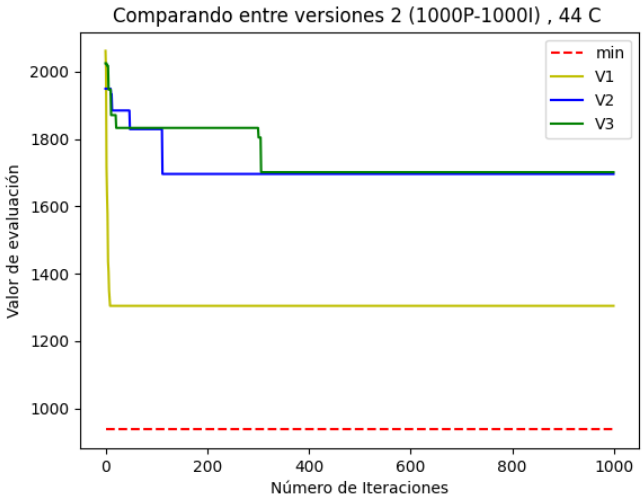


Figura 5: Comparación entre versiones para 1000 partículas, 1000 iteraciones y 44 clientes

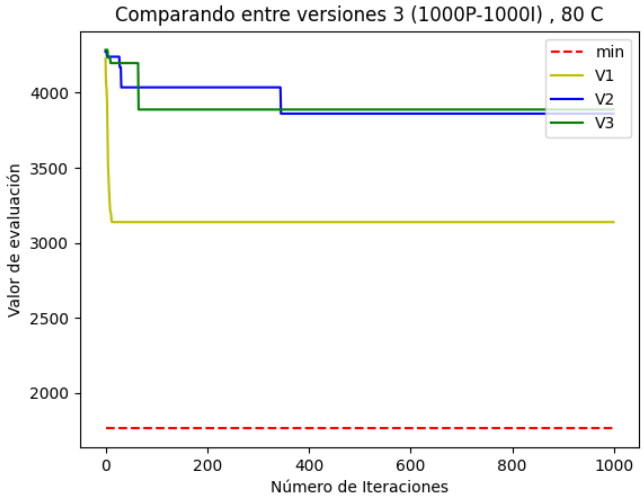


Figura 6: Comparación entre versiones para 1000 partículas, 1000 iteraciones y 80 clientes