An improved Particle Swarm Optimization Algorithm for the VRP with Simultaneous Pickup and Delivery and Time Windows

Carolina Lagos, Guillermo Guerrero, Enrique Cabrera, Andrés Moltedo, Franklin Johnson, Fernando Paredes.

***Abstract*— In logistics, planners' main goal is to reduce operational cost as much as possible. Keeping this in mind, other aspects such as recycling arise as important issues for customers. As a consequence, planners often need to find a balance among all these aspects and operational costs. In this work the vehicle routing problem with simultaneous pick-up and delivery and time windows is considered. Simultaneous pick-up and delivery (also called *reverse logistics*) allows us to address the problem of removing goods after they have been labelled as obsolete. To solve this complex combinatorial optimisation problem we use Particle Swarm Optimisation (PSO), a social-inspired algorithm. PSO aims to find a set of paths that minimises the total distance of the paths while serving, simultaneously, customers delivery and pick-up demands. Further, time windows constraints are also considered in this paper, which make the problem harder to solve. Including time windows makes also the problem more realistic, though. Results show that the PSO algorithm can find solutions that are quite competitive w.r.t. previously reported algorithms in literature. Furthermore, the PSO algorithm solves the problem within an acceptable time.**

***Keywords*—Particle swarm optimization, Vehicle Routing, Logistics, Heuristic Algorithm, Optimization.**

I. Introducción

La distribución de bienes es un elemento clave en logística. Por ello, se deben tomar en consideración varios aspectos tales como el número, tipo y/o capacidad de los vehículos de la flota así como también otros requerimientos específicos que pudieran venir desde el cliente. El problema de distribución de bienes a través de una flota de vehículos (*vehicle routing problem,* VRP) ha sido ampliamente estudiado en la literatura y, por lo mismo, nuevos elementos han sido incorporados en el modelo tradicional del VRP con el fin de cumplir con los nuevos requerimientos de la industria. Una de estas variaciones al modelo original VRP es conocido como *logística reversa* o de *entrega y recogida simultanea* (*simultaneous pick up and delivery, VRPSPD* [1]).

La entrega y recogida simultánea es un aspecto importante que debiera ser considerado cuando se modela el problema de distribución de bienes, ya que el hecho de retirar aquellos bienes etiquetados como *obsoletos* mientras se entrega el nuevo producto agrega valor al servicio entregado a los clientes.

Tal como el VRP, el VRPSPD es un problema de optimización combinatoria muy difícil de resolver [1-5]. Por esta razón, en la medida que el tamaño del problema aumenta (es decir el número de variables de decisión es más grande) los métodos de programación exacta tienden a fallar, puesto que no son capaces de encontrar la solución óptima para dichas instancias del problema. Como consecuencia de lo anterior, algunos métodos heurísticos han sido propuestos en la literatura para resolver el problema de manera aproximada, es decir, para obtener soluciones de buena calidad, pero para las cuales no hay garantías de optimalidad (en la Sección II de este artículo se presenta una breve revisión de estos métodos).

En esta investigación se busca encontrar soluciones de buena calidad para el VRPSPD considerando, adicionalmente, ventanas de tiempo, es decir, se agregan al problema original una serie de restricciones que aseguran que los clientes serán atendidos dentro de un rango de tiempo pre-establecido. Este problema es conocido como VRPSPD con ventanas de tiempo (VRPSPDTW). El modelo matemático y su explicación en detalle se incluyen en la siguiente sección de este trabajo.

El heurístico escogido para resolver este problema es enjambre de partículas (*Particle Swarm Optimization,* PSO). Una de las razones para la utilización de esta técnica es su eficiencia para resolver problemas de ruteo. En parte esto se debe a que es una técnica socio-inspirada, es decir donde varios individuos interactúan sin un control o guía central, lo que mejora la capacidad de exploración del algoritmo. Dentro de este tipo de técnicas también encontramos, por ejemplo, a los algoritmos de colonias de hormigas, que también han sido usados de manera satisfactoria en problemas de ruteo. En la Sección III de este trabajo se presenta una breve descripción de PSO. En la Sección IV se presentan los experimentos realizados y los resultados obtenidos. Finalmente en la última sección, las conclusiones del trabajo son presentadas y algunas líneas de investigación futura son propuestas.

II. El VRPSPDTW: Revisión Bibliográfica y Modelado

Esta sección comienza por presentar una vista general del VRPSPDTW. A continuación, la formulación matemática que se usa en este trabajo para modelar el VRPSPDTW es presentada y explicada en detalle.

En la literatura es posible encontrar muchos artículos de investigación que buscan resolver problemas de enrutamiento

C. Lagos, G. Guerrero, Escuela de Ingeniería Informática, Pontificia Universidad Católica de Valparaíso, 2362807, Chile, E. Cabrera, CIMFAV, Universidad de Valparaíso, 2362905, Chile, A. Moltedo, Escuela de Psicología, Universidad Santo Tomás, 2561694, Chile, F. Johnson, Departamento de Computación e Informática, Universidad de Playa Ancha, 2360003, Chile, F. Paredes, Departamento de Ingeniería Industrial, Universidad Diego Portales, 8370109, Chile.
Corresponding author: C. Lagos (carolina.lagos.c@mail.pucv.cl)

de vehículos y/o distribución de bienes (ver [6] para un estudio más exhaustivo de estos trabajos). Desde el primer trabajo académico en donde se considera el VRP a finales de los años 50s [7], muchos problemas de optimización basados en la versión original del VRP han sido propuestos. Uno de esos problemas es el VRPSPD. En las últimas tres décadas, el VRPSPD ha atraído la atención tanto de la academia como de la industria, y distintos enfoques han sido propuestos para tratar de resolver este problema. A modo de ejemplo, los autores en [1] estudiaron un caso real de logística reversa en la industria de distribución de libros. En ese estudio, el número de vehículos en la flota era fijo y cada uno de ellos tenía una capacidad limitada. Los autores proponían resolver el problema a través de un heurístico basado en inserciones, muy común para este tipo de problemas. Una pequeña variación al VRPSPD es la que se presenta en [8] donde la condición de simultaneidad no es requerida. En [8] los autores evitan el uso de heurísticos basados en inserción y, en su lugar, proponen una estrategia que considera la resolución del problema relajado (el cual reduce a VRP) y luego una fase de "ajuste" es llevada a cabo en donde las rutas obtenidas durante la primera fase son modificadas con el fin de asegurar que la solución final cumpla con las restricciones del VRPSPD.

Muchos autores han presentado diversas estrategias y marcos de trabajo para resolver, de manera aproximada, el VRPSPD. En [9], los autores propusieron una búsqueda local adaptativa que combina recocido simulado (*simulated annealing*) y búsqueda en vecindarios variables (*variable neighbourhood search,* VNS [10]) para obtener soluciones de buena calidad para el problema. En [11] un algoritmo de búsqueda tabú es combinado con una búsqueda local guiada. Este algoritmo híbrido busca lograr un balance en las habilidades de exploración del espacio de búsqueda y de intensificación de la búsqueda en zonas "promisorias" del espacio de búsqueda. La búsqueda tabú también es considerada en [12] para resolver una variante del VRPSPD que incluye restricciones que limitan la distancia máxima que cada vehículo puede recorrer.

Por otra parte, en [13] se implementa un algoritmo genético para el VRPSPD con múltiples objetivos. En ese trabajo, los costos de transporte y el número de vehículos necesarios con considerados como objetivos en conflicto. En [14] los autores proponen una mejora al algoritmo de evolución diferenciada (*differential evolution*). Los autores aseguran que su algoritmo se comporta mejor que los algoritmos evolutivos previamente propuestos en la literatura para el VRPSPD.

PSO también ha sido considerado para resolver problemas de enrutamiento de vehículos. Por ejemplo, en [29] los autores combinan PSO con un algoritmo de búsqueda local para resolver un problema de enrutamiento que considera una flota homogénea y ventanas de tiempo. A diferencia de lo que se hace en este artículo, los autores en [29] proponen usar una lista de prioridad de clientes. En [30] un algoritmo PSO que usa múltiples estructuras de aprendizaje social para resolver el VRPSPD sin ventanas de tiempo es presentado. Los autores no reportan el uso de estrategias de reparación. En [31] los autores aplican un algoritmo PSO a un problema de enrutamiento que considera demandas estocásticas. El algoritmo PSO usa una novedosa estructura de vecindario que, según los autores, puede ser aplicada a cualquier problema de enrutamiento. Aunque dicha estructura de vecindario difiere de la aplicada en este artículo, igualmente podría ser considerada en futuras extensiones de este trabajo. En [32] los autores proponen el uso de PSO para resolver un problema de enrutamiento con flota heterogénea y entrega y recogida simultánea. En su artículo, los autores destacan que el algoritmo PSO propuesto difiere de los algoritmos clásicos tanto en la representación de las partículas como en las operaciones aplicadas durante la ejecución del algoritmo. En general, una característica destacada de PSO es su rápida convergencia [3]. Otros algoritmos inspirados en enjambres también han sido aplicados al VRPSPD (por ejemplo [15]). Por otro lado, en [16] los autores presentan un conjunto de límites para el problema que permiten estimar la calidad de las soluciones entregadas por los algoritmos heurísticos.

Así como muchos algoritmos heurísticos han sido usados para encontrar buenas soluciones al VRPSPD, varios métodos exactos también han sido propuestos. Por ejemplo, en [17] los autores proponen un algoritmo basado en cortes (*cutting-plane-based*). En ese estudio, los autores destacan que su algoritmo es capaz de encontrar soluciones óptimas para instancias medianas del VRPSPD. En [18] los autores presentan un algoritmo de ramificación y poda (*branch-and-bound*) que resuelve el VRP con ventanas de tiempo y un número fijo de vehículos. Desafortunadamente, en la medida que el problema se hace más grande, los métodos exactos se vuelven poco prácticos ya que requieren mucho tiempo y memoria para poder obtener soluciones óptimas al problema.

Como se dijo anteriormente, el problema que se obtiene al agregar restricciones de tiempo al VRPSPD es llamado VRPSPDTW. Aunque menos estudiado que el VRPSPD, el VRPSPDTW es muy importante en logística reversa. En [19] los autores han propuesto un algoritmo PSO cooperativo con múltiples enjambres para resolver el VRPSPDTW. Este algoritmo se muestra mejor que algoritmos genéticos y el PSO tradicional. Otro algoritmo basado en PSO es propuesto en [20]. En ese trabajo los autores combinan PSO y VNS. Mientras PSO es usado para explorar el espacio de búsqueda, VNS es usado para mejorar los mejores individuos del enjambre. Otras estrategias hibridas para VRPSPDTW son presentadas en [21, 22, 23]. En un artículo recientemente publicado [24], se propone un algoritmo de colonia de hormigas (*ant colony optimisation,* ACO) para el VRPSPDTW. En ese artículo, ACO demostró ser muy competitivo para este problema, en especial cuando se compara con otras estrategias evolutivas. Finalmente, algunos algoritmos exactos para resolver el VRPSPDTW han sido propuestos. Por ejemplo, *branch-and-price* es considerado en [25]. En ese artículo, el VRPSPDTW es modelado como un problema de cobertura de conjuntos (*set covering problem*). Los autores afirman que este es el primer algoritmo exacto que es capaz de resolver el VRPSPDTW, aunque sólo instancias pequeñas (20 clientes) son reportadas.

Respecto del modelamiento del VRPSDP, este puede ser presentado de distintas maneras. En este artículo en particular, se considera un modelo para el VRPSPDTW que está basado en el modelo presentado en [2] y que es también usado en [24]. En la Tabla 1 se muestran los parámetros del problema que serán considerados en este artículo. Estos valores corresponden al tamaño de las instancias que serán consideradas en este artículo.

TABLA I
PARÁMETROS DEL VRPSPDTW

Parámetros					**Valor**		
# Máximo de vehículos (m)					10;25		
Capacidad vehículos (L)					[100,350]		
Número de clientes (n)					25;50;100		
v	x	y	d	p	t_{min}	t_{max}	t_s
0	x_0	y_0	--	--	--	--	--
1	x_1	y_1	d_1	p_1	t_{min}^1	t_{max}^1	t_s^1
...	...	...	...	...			
N	x_n	y_n	d_n	p_n	t_{min}^n	t_{max}^n	t_s^n

Como se puede ver en la Tabla I, en el VRPSPDTW un conjunto de clientes C debe ser atendido por una flota de vehículos V. Cada cliente $i \in C$ tiene su propia demanda de items que deben ser entregados y recogidos. Estas demandas se representan por d_i and p_i, respectivamente, donde $i = 1, \dots, n$, es el índice de los clientes en C. Se asume que tanto la entrega como la recogida de bienes la realiza el mismo vehículo $v = 1, \dots, m$ de manera simultánea. Un vehículo $v \in V$ puede atender uno o más clientes y siempre comienza y termina en un depósito central, 0. El problema, entonces, consiste en encontrar el conjunto de rutas para los vehículos de la flota que minimice la distancia total recorrida por los vehículos y que asegure, al mismo tiempo, que todos los clientes serán atendidos dentro de las ventanas de tiempo exigidas. Cada ruta es representada por una variable de decisión binaria x_{ijv} tal que $x_{ijv} = 1$ si el cliente *j* es visitado inmediatamente después del cliente i por el vehículo v y $x_{ijv} = 0$ si esto no es así.

Las restricciones asociadas a las ventanas de tiempo que se consideran en este trabajo aseguran que el cliente *i* sólo puede ser atendido durante el rango $[t_{min}^i, t_{max}^i]$. Cada visita al cliente i toma t_s^i unidades de tiempo y no depende del vehículo que realiza la atención.

Otra restricción del VRPSPDTW es que la capacidad del vehículo jamás puede ser superada. Para esto, primero se define la carga inicial del vehículo v antes de atender al primer cliente. Ésta carga inicial es expresada como

$$l_0^v = \sum_{i=0}^{n} \sum_{j=1}^{n} x_{ijv} d_j \tag{1}$$

Una vez que el primer cliente ha sido visitado, la carga del vehículo después de visitar a un cliente es

$$l_j^v \geq \begin{cases} l_0^v - d_j + p_j \forall j \in C, v \in V; if x_{0jv} = 1 \\ l_i^v - d_j + p_j \forall i, j \in C, v \in V; if x_{ijv} = 1 \end{cases} \tag{2}$$

La Ecuación (2) muestra que la carga *en ruta* del vehículo $v \in V$ es igual a la carga del vehículo después de atender al cliente i menos los items que son entregados al cliente j (d_j) más los ítems que son recogidos en el cliente j (p_j). Es claro que si

$$l_j^v > L, \tag{3}$$

la ruta se vuelve no factible. Para asegurar que todos los clientes son atendidos sólo una vez, se tiene que

$$\sum_{i=0}^{n} \sum_{v=1}^{m} x_{ijv} = 1 \, \forall j \in C \tag{4}$$

Además, es necesario asegurar que la entrega y recogida es realizada por el mismo vehículo:

$$\sum_{i=0}^{n} x_{ihv} = \sum_{j=0}^{n} x_{hjv} \, \forall h \in C, v \in V \tag{5}$$

Por otro lado, todos los vehículos deben comenzar la ruta en el depósito central

$$\sum_{j=1}^{n} x_{0jv} = 1 \, \forall v \in V \tag{6}$$

En relación a las ventanas de tiempo, se consideran tres restricciones:

$$st^j \geq \begin{cases} st^i + t_s^i + T_{ij} \forall i, j \in C, v \in V, si x_{ijv} = 1 \\ 0, sino \end{cases} \tag{7}$$

$$st^j \leq t_{max}^j \forall j \in C \tag{8}$$

$$st^j \geq t_{min}^j \forall j \in C, \tag{9}$$

donde st^j es el tiempo de inicio de la ventana de tiempo para atender al cliente j, y T_{ij} es el tiempo de viaje entre los clientes i and j. La Ecuación (7) asegura que hay tiempo suficiente para atender al cliente i y luego viajar hasta el próximo cliente j. La Ecuación (8) asegura que la atención del cliente *j* comenzará antes de t_{max}^j. Asimismo, la Ecuación (9) asegura que la atención del cliente *j* no empezará antes de t_{min}^j.

El problema VRPSPDTW que se considera en este artículo es, entonces:

$$min \sum_{i=0}^{n} \sum_{j=0}^{n} \sum_{v=1}^{m} D_{ij} x_{ijv}, \tag{10}$$

Sujeto a las Ecuaciones (1) a la (9).

En la siguiente sección se describe el algoritmo PSO considerado en este artículo. Se debe dejar en claro que el objetivo de este artículo es presentar nuestro algoritmo PSO mejorado para la resolución del problema VRPSPDTW, presentado anteriormente, y que en ningún caso se pretende hacer mejoras sobre el modelo del problema.

III ALGORITMO PSO MEJORADO

En este trabajo de investigación se propone un algoritmo PSO mejorado para la resolución del VRPSPDTW en (10). En esta sección las principales características del algoritmo propuesto son presentadas.

PSO fue por primera vez presentado en [26]. Es considerado un algoritmo socio-inspirado ya que trata de replicar el comportamiento social de grupos de animales tales como bandadas de aves o cardúmenes de peces. En general, PSO puede ser descrito como sigue: Una *partícula* φ es una solución del problema (no necesariamente la óptima). Un conjunto de particular, $\varphi \in \Phi$ (el cual es llamado *enjambre* o *swarm*) es inicializado de manera aleatoria. Todas las partículas tienen su propia *posición* y *velocidad*. Cada partícula se *mueve* por medio de un cambio en su velocidad y/o posición. Además, cada partícula lleva un registro de las posiciones en las que estuvo anteriormente. Por ello, cada

partícula conoce la *mejor posición* en la que ha estado, es decir, la posición en la que alcanzo su mejor valor de *fitness*. Esta experiencia es tomada en cuenta al momento de influenciar la próxima posición y velocidad de la partícula. Esta experiencia es también compartida con el resto del enjambre. La mejor partícula del enjambre es identificada como la *solución global*. La posición y la velocidad de la solución global también son consideradas por el resto de la población para definir su nueva posición.

La posición de la partícula φ en la siguiente iteración es calculada como

$$\varphi_i^{k+1} = \varphi_i^k + v_i^{k+1} \cdot \Delta t \tag{11}$$

donde,

$$v_i^{k+1} = \omega \cdot v_1^k + c_1 \cdot r_1^k\left(\varphi_i^p - \varphi_i^k\right) + c_2 \cdot r_2^k\left(\varphi_n^g - \varphi_i^k\right). \tag{12}$$

En la Ecuación (11), Δt es el tiempo durante el cual la velocidad v^{k+1} es aplicada sobre la posicion actual de la partícula φ^k. En este paper ese valor es fijado en 1. En la Ecuación (12), c_1 y c_2 son constantes de aceleración, también llamados *tendencia social*. El parámetro $\omega \in [0,1]$ el peso inercial. Parámetros r_1^k and r_2^k son números aleatorios uniformemente distribuidos dentro del intervalo [0,1]. La mejor posición que una partícula φ alcanzó durante las iteraciones pasadas es φ . Luego, φ_i en la Ecuación (12) corresponde a la i-ésima coordenada de φ . Del mismo modo, la mejor posición alcanzada por el enjambre Φ (*solución global*) se representa por φ^g. Como se muestra en la Ecuación (12), las mejores posiciones (global e individuales) influencian la velocidad que será aplicada a las partículas en la siguiente iteración.

Una mejora que se hace en este trabajo al PSO original es fijar φ como la mejor posición local. Esto es, se define un vecindario $N \subset \Phi$ y luego se comparan aquellas partículas dentro de ese vecindario. La mejor posición local se determina como

$$\varphi = \begin{cases} best(j \in N \subset \Phi(\varphi^j) \cup \{\varphi^i\}), si\ r < R \\ \varphi\ , sino \end{cases} \tag{13}$$

Es importante notar que el vecindario de cada partícula variará en cada iteración ya que el vecindario es definido por la posición actual de cada partícula. El tamaño del vecindario es definido por el usuario. El parámetro $0 < R < 1$ en Ecuación (13) es también definido por el usuario.

Uno de los elementos claves para la correcta aplicación del PSO es la representación de la solución al problema que se quiere abordar. Es importante que la representación usada permita explotar las características particulares del problema de optimización que se quiere resolver. En este trabajo, cada partícula es representada por un vector, como se muestra en la Fig. 1.

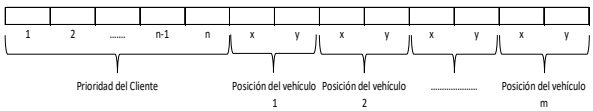


Figura 1. Representación de una partícula.

Los primeros n elementos de una partícula corresponden a la prioridad de cada cliente i. La prioridad inicial de cada cliente es un valor aleatorio. Los últimos $2m$ elementos de una partícula corresponden a las coordenadas de los m vehículos de la flota. Se necesitan 2 elementos por vehículo para poder representar las coordenadas (x, y) de su posición en el espacio Euclidiano. La posición inicial de cada vehículo también es un número aleatorio. Luego, los $(n + 2m)$ elementos de una partícula definen su posición. Esta representación nos permite mantener a las partículas en el espacio continuo, lo cual favorece el buen rendimiento de PSO, el cual es un algoritmo que fue concebido para trabajar con variables continuas.

A continuación se muestra, a través de un ejemplo, como el algoritmo PSO propuesto en este artículo funciona: Asumamos que tenemos 6 clientes y 3 vehículos disponibles. Entonces, cada partícula será representada por un vector de largo 12, como el que se muestra en Fig. 2.

0.3	0.7	0.1	0.9	0.2	0.4	35	44	12	95	66	89
1	2	3	4	5	6	v_1^x	v_1^y	v_2^x	v_2^y	v_3^x	v_3^y

Figura 2. Ejemplo de una partícula según la estructura usada en este artículo.

Los valores del vector en Fig. 2 han sido generados de manera aleatoria. Las coordenadas de la posición de cada vehículo deben estar dentro de un cuadrado imaginario de $[100 \times 100]$ unidades de distancia. PSO genera un gran número de partículas como la que se muestra en Fig. 2. Este conjunto de partículas es llamado enjambre o *swarm*.

En cada partícula del enjambre, los clientes son ordenados de acuerdo a sus prioridades. Luego, una matriz de preferencia es construida, donde por cada cliente los vehículos son ordenados de acuerdo con su cercanía con el cliente. La Tabla II muestra un ejemplo de esta matriz. Para el ejemplo, el vehículo más cercano al cliente 1 es el vehículo número 3. Si el vehículo 3 no puede atender al cliente 1 (debido a restricciones horarias y/o de capacidad), entonces el segundo vehículo más cercano es evaluado (para el ejemplo, esto sería el vehículo numero 1). Si, de nuevo, el vehículo 1 no puede atender al cliente 1, entonces el vehículo 2 es evaluado.

TABLA II
VEHÍCULOS ORDENADOS SEGÚN LA PREFERENCIA DE CADA UNO DE LOS CLIENTES.

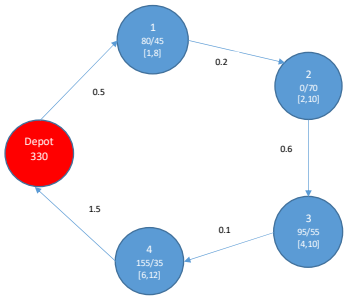
Clientes	Vehículos según preferencia		
1	3	1	2
2	2	1	3
3	2	1	3
4	3	2	1
5	1	2	3
6	3	2	1

Si ningún vehículo puede atender al cliente 1, entonces el cliente 1 es asignado al vehículo con la mayor capacidad de carga disponible. Claramente, hacer esto lleva a la creación de rutas infactibles que deberán ser reparadas.

El proceso descrito más arriba se repite hasta que todos los

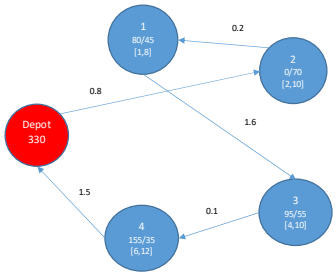
clientes son asignados a un vehículo. Luego de esto, se debe asegurar que las rutas que resultan de la asignación son factibles. En general, se observa que las rutas obtenidas inmediatamente después de la asignación inicial son infactibles. Por lo anterior una *fase de reparación* es necesaria. En este artículo, la estrategia de reparación propuesta en [28] es usada para adecuar el algoritmo PSO al problema en (1). Esta fase de reparación consiste de dos partes. En la primera parte, los clientes asignados a un vehículo son ordenados de acuerdo con su disponibilidad horaria (ventanas de tiempo). Aquellos clientes que pueden ser atendidos más temprano son asignados al comienzo de la ruta, mientras que aquellos clientes que deben ser atendidos más tarde son puestos al final de la ruta. La Fig. 3(a), muestra un ejemplo de esta situación. Si la ruta obtenida sigue siendo infactible, una secuencia de movimientos del tipo *2-opt* es aplicada. Estos movimientos son realizados entre aquellos clientes "conflictivos", es decir, clientes que están violando alguna restricción de tiempo y/o de capacidad (ver Fig. 3(b) y 3(c)).

En la mayoría de los casos, luego de aplicar la secuencia de movimientos *2-opt* las rutas se vuelven factibles. Sin embargo, en algunos casos es necesario pasar a la segunda parte de la fase de reparación.



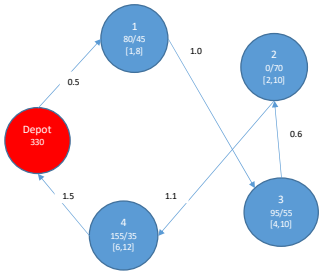
Vehículo	Carga	t_min
depot	330	0
1	295	1
2	365	2
3	325	4
4	205	6

(a) Ejemplo de una ruta para un vehículo. Los clientes son ordenados según sus ventanas de tiempo (entre los paréntesis "[ ]"). Luego de visitar al cliente 2 (en segundo lugar) la restricción de capacidad es violada (≤ 350), lo que vuelve la ruta infactible.



Vehículo	Carga	t_min	
depot	330	0	
2	400	2	*
1	365	3.2	*
3	325	5.8	
4	205	6.9	

(b) Ejemplo de una ruta para un vehículo – fase de reparación (parte 1). Se realiza un movimiento del tipo *2-opt* entre los clientes 1 y 2. La ruta obtenida luego del movimiento sigue siendo infactible y por lo tanto se descarta.



Vehículo	Carga	t_min
depot	330	0
1	295	1
3	255	4
2	325	5.6
4	205	7.7

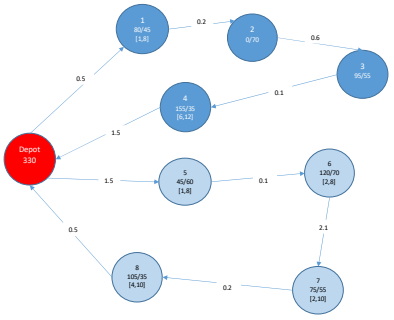
(c) Ejemplo de una ruta para un vehículo – fase de reparación (parte 1).

Nuevamente, un movimiento tipo 2-opt es aplicado entre los clientes 2 y 3. La ruta obtenida es factible y por lo tanto no se requiere ejecutar la parte 2 de la estrategia de reparación para este vehículo.

Figura 3. Ejemplo de la primera parte de la estrategia de reparación. A partir de movimientos tipo *2-opt* es posible reparar una ruta infactible y volverla factible.

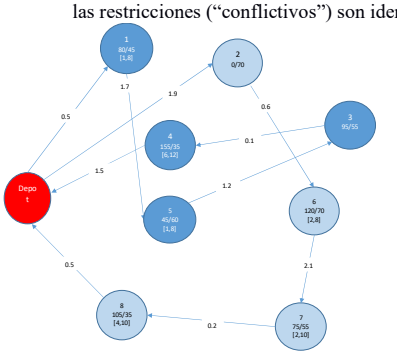
En la segunda parte de la fase de reparación, aquellos clientes conflictivos de las rutas infactibles son intercambiados. Normalmente después del intercambio las nuevas rutas siguen siendo infactibles.

Por lo anterior, luego del intercambio, la secuencia de movimientos *2-opt* de la primera parte de la fase de reparación se vuelve a aplicar. Fig. 4 shows an example of the second stage of the restoration phase.



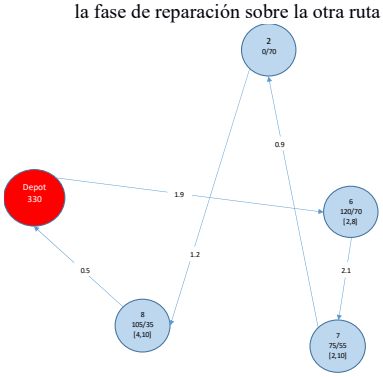
Vehículo	Carga	t_min	
depot	330	0	
1	295	1	
2	365	2.2	*
3	325	4	
4	205	6	
depot	345	0	
5	360	1.5	*
6	310	2.6	
7	290	5.7	
8	220	6.9	

(a) Ejemplo de dos rutas infactibles. Los clientes que no cumplen con las restricciones ("conflictivos") son identificados (clientes 2 y 5).



Vehículo	Carga	t_min	
depot	330	0	
1	295	1	
5	310	3.7	
3	270	5.9	
4	150	7	
depot	345	0	
2	415	2	*
6	365	3.6	*
7	345	5.8	
8	275	7	

(b) Ejemplo de dos rutas infactibles – fase de reparación (parte 2). Un movimiento tipo *2-opt* es aplicado entre los clientes identificados como conflictivos. (2 y 5). Una de las rutas se vuelve inmediatamente factible después del cambio. Se repite la parte 1 de la fase de reparación sobre la otra ruta que se mantiene infactible.



Vehículo	Carga	t_min
depot	345	0
6	295	1.9
7	275	5
2	345	6.9
8	275	9.1

(c) Ejemplo de dos rutas infactibles – fase de reparación (parte 1). Dos movimientos tipo *2-opt* son ejecutados. Primero los clientes 2 y 6 intercambian posiciones. Aunque la ruta sigue siendo infactible después de este movimiento, la nueva ruta no se descarta ya que el cliente "conflictivo" (originalmente 6) deja de serlo. Luego, un segundo movimiento entre los clientes 2 y 7 es realizado. La ruta obtenida se vuelve factible.

Figura 4. Ejemplo de la segunda parte de la fase de reparación. A través de movimientos tipo *2-opt* entre clientes de distintas rutas se logra repararlas y

obtener una solución factible al problema.

En nuestra experiencia, luego de un par de iteraciones de la fase de reparación, todas las rutas son factibles. Una vez que se obtienen sólo rutas factibles para el problema en (10), el costo de dicha solución es calculado.

Algoritmo 1: Particle Swarm Optimisation

```
Begin
  Φ ←InitialiseSwarm();
  While (terminationCriteria= False)
    For each φ ∈ Φ
      φ^fitness=computeFitness(φ)  // Fase de reparación es embebida aquí
      If φ^fitness < φ  then
        φ = φ^fitness
      end
    end
    For each φ ∈ Φ
      φ = bestLocal(φ , N(φ), R)
    end
    φ^g = globalBest(Φ)
    For each φ ∈ Φ
    φ = updatePosition(ω, c1, c2, r1, r2, φ , φ^g )
    End
  End
End
```

IV. Experimentos

En esta sección se presenta, en primer lugar, el conjunto de instancias que se usará en los experimentos de este artículo. Luego, los resultados obtenidos por el algoritmo PSO son presentados y discutidos.

Como se propone en [25], el conjunto de instancias que se usará para testear nuestro algoritmo está basado en las instancias propuestas por Solomon para el VRPTW [27]. Aquí, se asume que los valores entregados por Solomon corresponden a las demandas de entrega d_i, $i = 1, \dots, n$. Las demandas de recogida p_i, $i = 1, \dots, n$, son calculadas como sigue:

$$p_i := \begin{cases} (1-\alpha)d_i, siiespar; \\ (1+\alpha)d_i, siiesimpar; \end{cases}$$

donde $0 \leq \alpha \leq 1$. El conjunto de instancias resultantes es similar al de [24].

Los parámetros usados para el PSO implementado en este artículo son indicados en la Tabla III.

TABLA III.
Parámetros del Algoritmo PSO

Parámetro	Valor
Iteraciones	1500
# Partículas	100
Peso Inercial (w)	0.75
Velocidad Max Min	0.25
Posición Max	1.00
Posición Min	0.00
C1	1.49
C2	1.49

Los valores de los parámetros en la Tabla III fueron obtenidos después de un proceso de ajuste basado en un enfoque de prueba y error y, por lo tanto, podrían no ser los "mejores" valores para dichos parámetros.

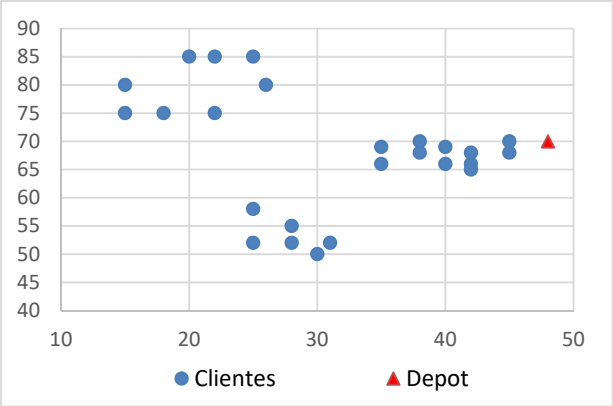


Figura 5. Ejemplo de una instancia en donde los clientes (círculos) están distribuidos en clusters.

La Fig. 5 muestra una de las instancias que se consideran en este artículo. Los círculos representan la ubicación de los clientes, mientras que los triángulos representan la ubicación de los depósitos. Para la instancia de la Fig. 5 es posible notar que los clientes están distribuidos en 3 clusters. Dentro de las instancias consideradas en este estudio, también existen algunas en donde los clientes están uniformemente distribuidos en el espacio (sin clusters).

Como se puede ver en la Tabla IV, PSO es capaz de encontrar una solución al VRPSPDTW en (1) para todas las instancias testeadas en este trabajo. Además, el número de vehículos requeridos por las soluciones encontradas por PSO, m, es siempre menor que el máximo permitidoV^{max}. Por otra parte, en relacion al tiempo requerido por PSO para converger a una solución para el VRPSPDTW en (1) es posible notar que PSO converge más rápidamente que otros algoritmos presentes en la literatura. En particular, en este trabajo comparamos los resultados obtenidos por nuestro PSO con el algoritmo de hormigas presentado en [24]. Los resultados obtenidos por nuestro algoritmo PSO también mejoran algunos de los resultados obtenidos por los algoritmos de búsqueda local recientemente presentados en [28] y que utilizan la misma estrategia de reparación que utilizamos en este trabajo. Dado que en [24] sólo instancias con 50 clientes fueron testeadas, para efectos de la comparación sólo se consideraran las instancias Spdtw8 a la Spdtw15. Los resultados muestran que el algoritmo PSO presentado en este artículo reduce en casi un 5% el tiempo que los algoritmos de hormiga en [24] necesitan para resolver el problema. Los experimentos fueron realizados en la misma máquina usada en [24] y [28].

En la Fig. 6 se muestran las rutas de los 6 vehículos que se necesitan para resolver la instancia Spdtw1 de la Tabla IV. Es posible notar que los vehículos no necesariamente visitan a los clientes más cercanos a su posición actual. Esto se debe a que, en algunos casos, las restricciones de capacidad y/o tiempo obligan a visitar otros clientes que están más lejos antes de visitar aquellos que están más cerca en términos de distancia Euclidiana. A pesar de lo anterior, los vehículos tienden a mantenerse dentro del mismo cluster de clientes. Se observa también que, para algunas pocas instancias, las rutas de los vehículos incluyen clientes que pertenecen a clusters distintos.

TABLA IV
RESULTADOS OBTENIDOS POR EL ALGORITMO PSO

Instancia	V^{max}	n	m	L^{max}	f	t
Spdtw1	10	25	6	200	237.85	28
Spdtw2	10	25	4	350	339.28	29
Spdtw3	10	25	6	350	333.52	25
Spdtw4	10	25	4	350	279.21	18
Spdtw5	10	25	8	100	443.78	20
Spdtw6	10	25	9	100	553.51	18
Spdtw7	10	25	6	160	349.59	21
Spdtw8	25	50	8	200	1112.52	33
Spdtw9	25	50	11	200	744.01	51
Spdtw10	25	50	11	200	2315.75	52
Spdtw11	25	50	14	350	2161.63	81
Spdtw12	25	50	9	350	1601.98	71
Spdtw13	25	50	15	100	2624.12	88
Spdtw14	25	50	15	100	2092.07	87
Spdtw15	25	50	6	160	919.52	45
Spdtw16	25	100	24	200	46.75.51	182
Spdtw17	25	100	22	200	3991.77	163
Spdtw18	25	100	15	350	3612.56	141

Lo anterior se debe, principalmente, a la segunda parte de la etapa de reparación implementada en este trabajo.

La Fig. 7 muestra las rutas obtenidas por el algoritmo PSO cuando la capacidad de los vehículos en la instancia Spdtw1 es aumentada hasta 350.

En este caso, sólo 4 vehículos son necesarios para completar todas las entregas y recogidas. Nuevamente, todas las rutas incluyen solo clientes que pertenecen al mismo cluster. Como era esperable, mientras más aumenta la capacidad del vehículo, éste es capaz de atender más clientes durante la ruta.

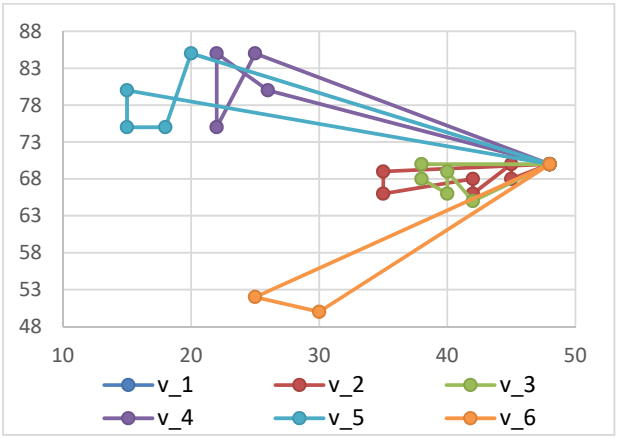


Figura 6. Rutas obtenidas para la instancia Spdtw1 con una capacidad total del vehículo igual a 200. Seis vehículos son requeridos en esta solución. (v_1 al v_6).

El orden en que los clientes son atendidos en este caso cambia con respecto al orden en que fueron atendidos antes de aumentar la capacidad de los vehículos (ver Fig. 6).

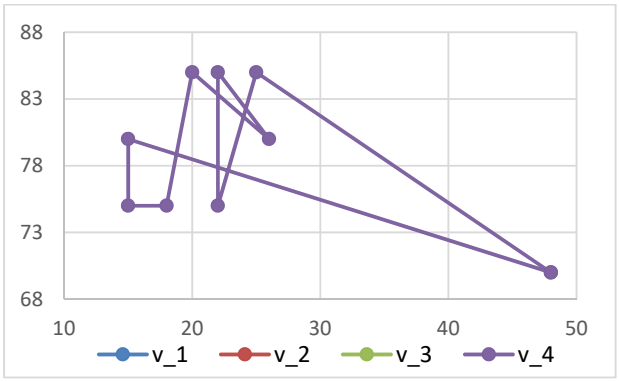


Figura 7. Rutas obtenidas para la instancia Spdtw1 después de aumentar la capacidad total del vehículo a 350. Sólo 4 vehículos son requeridos (v_1 to v_4).

Finalmente, en la Fig. 8 se muestra una instancia cuyos clientes están distribuidos de manera aleatoria uniforme. En este caso, se necesitan seis vehículos para resolver el problema (ver Fig. 9). En general, el hecho de considerar clientes distribuidos de manera uniforme aleatoria provoca un aumento en el número de vehículos respecto a instancias de igual tamaño pero con clientes organizados en clusters.

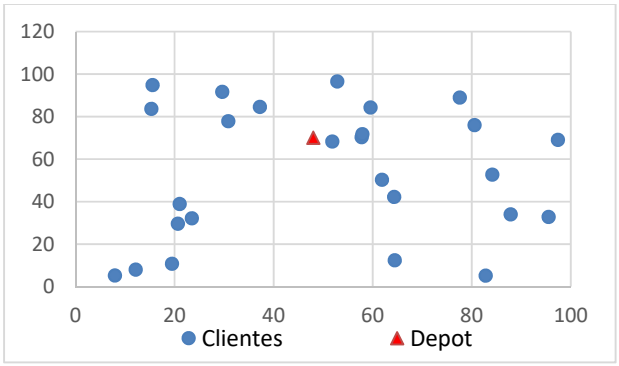


Figura 8. Ejemplo de una instancia con clientes distribuidos de manera aleatoria.

Finalmente, es interesante notar que las rutas de los vehículos en la solución para la instancia con clientes distribuidos de manera aleatoria muestran ciertos “clusters” que no son tan claros al momento de ver a los clientes sin las rutas, como se muestra en la Fig. 8.

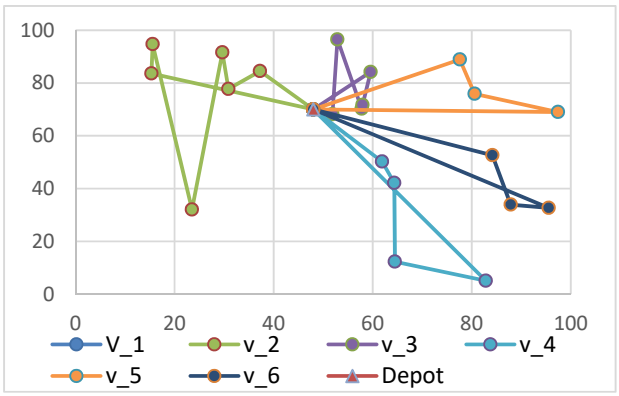


Figura 9. Rutas obtenidas para la instancia con clientes distribuidos de manera aleatoria. Seis vehículos son requeridos (v_1 to v_6).

V. CONCLUSIONES

En este artículo se implementó un algoritmo PSO el cual hace uso de una estrategia de reparación de rutas para resolver el problema de enrutamiento de vehículos con entrega y recogida simultánea con restricciones de tiempo. Una diferencia importante con otros algoritmos implementados para resolver problemas de enrutamiento es que la estrategia de reparación permite encontrar rutas factibles que no son tan distintas a aquellas generadas originalmente por el algoritmo y que resultaron ser infactibles. De esta manera, el funcionamiento general de PSO no se ve afectado por partículas en posiciones que son, en esencia, generadas al azar.

El algoritmo PSO implementado en este artículo demostró ser muy competitivo respecto de otros algoritmos socio-inspirados recientemente propuestos en la literatura. En particular, cuando se comparan los resultados obtenidos por nuestro algoritmo con los reportados en [24] (algoritmo de colonia de hormigas) se observa una reducción del tiempo que necesita el algoritmo para converger (PSO es 5% más rápido, usando la misma máquina para las pruebas).

REFERENCES

[1] H. Min. "The multiple vehicle routing problem with simultaneous delivery and pick-up points". Transportation Research Part A General 23(5), pp. 377-386, 1989.

[2] J. Dethloff, "Vehicle routing and reverse logistics: The vehicle routing problem with simultaneous delivery and pick-up". *OR-Spektrum*, vol. 23(1), pp. 79–96, 2001.

[3] T.J. Ai, V. Kachitvichyanukul. "A particle swarm optimization for the vehicle routing problem with simultaneous pickup and delivery". *Computers & Operations Research,* vol. 36(5), pp. 1693-1702, 2009.

[4] J. Fan, "The Vehicle Routing Problem with Simultaneous Pickup and Delivery Based on Customer Satisfaction", *Procedia Engineering*, vol. 15, pp. 5284-5289, 2011.

[5] J.F. Chen, T.H. Wu. "Vehicle routing problem with simultaneous deliveries and pickups". *Journal of the Operational Research Society*, vol. 57(5), pp 579–587, 2006.

[6] K. Braekers, K. Ramaekersa, I.V. Nieuwenhuysec, "The vehicle routing problem: State of the art classification and review". *Computers & Industrial Engineering*. doi:10.1016/j.cie.2015.12.007, to be published.

[7] G.B. Dantzig, J.H. Ramser. "The Truck Dispatching Problem". *Management Science,* vol. 6(1), pp. 80–91, 1959. doi:10.1287/mnsc.6.1.80

[8] G. Nagy, S. Salhi. "Heuristic Algorithms For Single And Multiple Depot Vehicle Routing Problems With Pickups And Deliveries". *European Journal of Operational Research*, vol. 162(1), pp. 126-141, 2005.

[9] M. Avci, S. Topaloglu. "An Adaptive Local Search Algorithm for Vehicle Routing Problem with Simultaneous and Mixed Pickups and Deliveries". Computers & Industrial Engineering, vol. 83(C), pp. 15-29, 2015. doi: 10.1016/j.cie.2015.02.002

[10] N. Mladenović, P. Hansen. "Variable neighborhood search". *Computers & Operations Research*, vol. 24(11), pp. 1097-1100, 1997.

[11] E. Zachariadis, C.D. Tarantilis, C. Kiranoudis. "A hybrid metaheuristic algorithm for the vehicle routing problem with simultaneous delivery and pick-up service". *Expert Systems with Applications*, vol. 36(2), pp. 1070-1081, 2009.

[12] F.A.Tang, R.D. Galvao. "A Tabu Search Algorithm for the Vehicle Routing Problem with Simultaneous Pick-Up and Delivery Service". *Computers & Operations Research*, vol. 33, pp. 595-619, 2006.

[13] N. Zhu, C. Shao. "Vehicle Routing Problem with Simultaneous Delivery and Pick-up Based on the Improved Genetic Algorithm" *in Proc. Fourth International Conference on Genetic and Evolutionary Computing (ICGEC)*, Shenzhen, pp.312 - 316, 2010.

[14] E. Cao, M. Lai. "An Improved Differential Evolution Algorithm for the Vehicle Routing Problem with Simultaneous Delivery and Pick-up Service", *in Proc. of the Third International Conference on Natural Computation (ICNC)*, pp. 436 - 440, 2007.

[15] Y. Gajpal, and P. Abad , "An ant colony system (ACS) for vehicle routing problem with simultaneous delivery and pickup", *Computers & Operations Research*, vol. 36(1), pp. 3215 – 3223, 2009.

[16] A. Subramanian ,and L. S. Ochi , "New Lower Bounds for the Vehicle Routing Problem with Simultaneous Pickup and Delivery", Technical Report - RT 01/10, Universidade Federal Fluminense, Niteri-RJ, Brazil, 2010.

[17] J. Rieck, J. Zimmermann, "Exact Solutions to the Symmetric and Asymmetric Vehicle Routing Problem with Simultaneous Delivery and Pick-Up". Business Research, vol. 6(1), pp. 77–92, 2013.

[18] A.W.J. Kolen, A.H.G. Rinnooy Kan, H.W.J.M. Trienekens. Vehicle Routing with Time Windows. *Operations Research*, vol. 35(2), pp. 266-273. 1987.

[19] X. Gan, Yan Wang, Shuhai Li, and Ben Niu. "Vehicle Routing Problem with Time Windows and Simultaneous Delivery and Pick-Up Service Based on MCPSO". *Mathematical Problems in Engineering*, vol. 2012, Article ID 104279, pp. 1-11, 2012. doi:10.1155/2012/104279

[20] F.P. Goksal, I. Karaoglan, F. Altiparmak. "A hybrid discrete particle swarm optimization for vehicle routing problem with simultaneous pickup and delivery". Computer and Industrial Engineering, vol. 65(1), pp. 39-53, 2013. doi: 10.1016/j.cie.2012.01.005.

[21] S. Kassem, M. Chen. "Solving reverse logistics vehicle routing problems with time windows". *The International Journal of Advanced Manufacturing Technology*, vol. 68(1), pp. 57–68, 2013. doi 10.1007/s00170-012-4708-9

[22] C. Wang, F. Zhao, D. Mu, J.W. Sutherland. "Simulated Annealing for a Vehicle Routing Problem with Simultaneous Pickup-Delivery and Time Windows". *Advances in Production Management Systems. Sustainable Production and Service Supply Chains*, vol. 415(2), pp 170-177, 2013.

[23] R. Liu, X. Xie, V. Augusto, C. Rodriguez. "Heuristic algorithms for a vehicle routing problem with simultaneous delivery and pickup and time windows in home health care". *European Journal of Operational Research*, vol. 230(3), pp. 475–486, 2013.

[24] F. Johnson, J. Vega, G. Cabrera, E. Cabrera, "Ant Colony System for a Problem in Reverse Logistic", *Studies in Informatics and Control*, vol. 24(2), pp. 133-140, 2015.

[25] E. Angelelli, R. Mansini. The Vehicle Routing Problem with Time Windows and Simultaneous Pick-up and Delivery. *Quantitative Approaches to Distribution Logistics and Supply Chain Management,* vol. 519 of the series Lecture Notes in Economics and Mathematical Systems, pp. 249-267, 2002.

[26] J. Kennedy, R. Eberhart. Particle swarm optimization, in *Proc. of the IEEE International Conference on Neural Networks*, vol. 4(1), pp. 1942-1948, 1995. doi: 10.1109/ICNN.1995.488968

[27] M. M. Solomon. "Algorithms for the Vehicle Routing Problem with Time Windows". *Transportation Science*, vol. 29(2), pp. 156-166, 1995.

[28] P. Cabrera-Guerrero, A. Moltedo-Perfetti, E. Cabrera, F. Paredes, "Comparing Two Heuristic Local Search Algorithms for a Complex Routing Problem", *Studies in Informatics and Control*, ISSN 1220-1766, vol. 25(4), pp. 411-420, 2016.

[29] F. Belmecheri, C. Prins, F. Yalaoui, L. Amodeo. "Particle swarm optimization algorithm for a vehicle routing problem with heterogeneous fleet, mixed backhauls, and time windows". *Journal of Intelligent Manufacturing*, vol. 24, pp. 775–789, 2013.

[30] R. Wei, T. Zhang, H. Tang. "An Improved Particle Swarm Optimization Algorithm for Solving Vehicle Routing Problem with Simultaneous Pickup and Delivery", *Technical Note*, 2010.

[31] Y. Marinakis. "An improved particle swarm optimization algorithm for the capacitated location routing problem and for the location routing problem with stochastic demands", *Applied Soft Computing*, vol. 37, pp. 680–701, 2015.

[32] B. Cassins, P. H. Siqueira, G.F. Aguiar, L. V. de Souza. "Particle Swarm Optimization for Vehicle Routing Problem with Fleet Heterogeneous and Simultaneous Collection and Delivery". Applied Mathematical Sciences, vol. 8(77), pp. 3833 – 3849, 2014.

Carolina Lagos obtuvo su licenciatura en ciencias de la ingeniería en 2006 en la PUCV (Chile). Actualmente se encuentra cursando su maestría en la misma universidad. Sus áreas de interés son la optimización combinatoria y la inteligencia artificial aplicada a problemas de políticas públicas.

Guillermo Guerrero obtuvo su Licenciatura en Ciencias de la Ingeniería en 2016 en la PUCV (Chile). Actualmente se desempeña como analista de la corporación municipal de Valparaíso. Sus áreas de interés son inteligencia artificial y algoritmos para problemas combinatorios complejos.

Enrique Cabrera obtuvo su licenciatura en matemáticas en 1972 en la PUCV (Chile) y su maestría en estadística en el CIENES de la Universidad de Chile en 1976. Actualmente es profesor adjunto de la Universidad de Valparaíso y es parte del CIMFAV. Sus áreas de interés son la inferencia, el análisis multivariado y la investigación de operaciones.

Andrés Moltedo-Perfetti obtuvo su título de Psicólogo en la PUCV (Chile). Posee dos grados de maestría: Universidad de La Laguna en 2003 (España) y Universidad de Valparaíso en 2007 (Chile). Obtuvo su doctorado en 2015 en la PUCV (Chile). Sus áreas de interés se centran en el estudio de los comportamientos sociales de grupos de personas, y su aplicación en diversas áreas científicas, tales como ingeniería, matemáticas y ciencias sociales.

Franklin Johnson obtuvo su licenciatura en ciencias de la ingeniería en 2006 en la PUCV. En 2014 obtuvo su maestría y en 2017 el grado de doctor en informática ambos de la PUCV. Actualmente es el director del departamento de computación de la Universidad de Playa Ancha. Sus áreas de interés son las metaheurísticas, búsqueda autónoma y la optimización combinatoria.

Fernando Paredes obtuvo su licenciatura en Matemáticas y su maestría en 1975, en la Universidad Santa María (Chile). Obtuvo su doctorado en ciencias en sistemas de ingeniería en el COPPE de la Universidad Federal de Rio de Janeiro (Brasil) en 1991. Sus áreas de interés son la investigación de operaciones y la optimización combinatoria.