EL PROBLEMA DE RUTEO DE VEHÍCULOS EN UNA EMPRESA PRODUCTORA DE ALIMENTOS

Francisco Ríos Molina

2018

UNIVERSIDAD ADOLFO IBÁÑEZ
FACULTAD DE INGENIERÍA Y CIENCIAS

EL PROBLEMA DE RUTEO DE VEHÍCULOS EN UNA EMPRESA PRODUCTORA DE ALIMENTOS

Francisco Ríos Molina

Profesor guía: Karol Suchan

TESIS PARA OPTAR AL TÍTULO DE INGENIERO CIVIL INDUSTRIAL CONCENTRACIÓN EN MINERÍA Y AL GRADO DE MAGISTER EN CIENCIAS DE LA INGENIERÍA – INGENIERÍA INDUSTRIAL

2018

ÍNDICE

GLOSARIO Y ABREVIATURAS

VRP: Vehicle routing problem.

CD: Centro de distribución.

TSP: Travelling salesman problem.

m-TSP: Multiple travelling salesman problem.

CVRP: Capacited vehicle routing problem.

VRPTW: Vehicle routing problem with time windows.

CVRPTW: Capacited vehicle routing problem with time windows.

SA_ Simulated Annealing

GRASP: Greedy randomized adaptive search procedure.

ConVRP: Consistent vehicle routing problem.

HORECA: Hoteles, restaurants y casinos.

ACO: Ant colony optimization.

I. RESUMEN EJECUTIVO

En la siguiente investigación se expone el caso de una Empresa productora de alimentos nivel nacional. Desde sus centros de distribución despachan pedidos a sus clientes a diario, esta demanda se recibe también diariamente y presenta estacionalidad semanal. Actualmente, el único indicio de planificación de ruta en la Empresa es la preferencia que manifiestan por atender a cada cliente siempre con el mismo camión. De todos modos, esta asignación es arbitraria y no responde a ningún criterio de optimización, por lo que, si bien se produce un servicio consistente, este no es óptimo. Esta situación genera ineficiencia en el servicio, ya que, aunque la flota de camiones con que cuenta la Empresa es capaz de satisfacer exitosamente los pedidos en días de baja demanda, no logra servir a todos los clientes en los días en que esta es alta.

Frente a este problema se realiza, en primer lugar, una optimización base de las rutas de entrega, que maximice los niveles de servicio y la cantidad de producto entregada. Para esto se modela la situación como un VRP, considerando como restricciones la capacidad de los vehículos y las ventanas de tiempo de los clientes. La resolución de este problema logra reducir en un 40% la distancia recorrida por la flota, junto con lo que aumenta en un 26% el nivel de servicio en los días de alta demanda y 27% la cantidad de producto entregada.

Luego, la consistencia es introducida al modelo, entendida como que cada cliente sea atendido por el mismo camión siempre. Sin embargo, el ConVRP está diseñado para planificar las rutas de una serie de días seguidos y en la Empresa no se conoce la demanda futura, por lo que se plantea incorporar incentivos a la consistencia en el ruteo en lugar de integrarla como una restricción. Lo que, de todos modos, refleja de manera adecuada la posición de la Empresa, para la cual la asignación de un mismo camión a cada cliente es una preferencia, no una necesidad.

Se formuló un primer modelo modificado, en que se incorpora una disminución en los tiempos de entrega, producto de los conocimientos adquiridos por el chofer al atender siempre al mismo cliente. Este modelo no produjo los resultados esperados, manteniendo un nivel de consistencia similar al modelo base, en donde solo el 12% de los clientes son servidos por el mismo chofer.

En una segunda modificación, se utilizan penalizaciones en la función objetivo, con lo que cuando un camión visita a un cliente que no tiene asignado, se suma un costo extra. Con esto se logra aumentar la consistencia hasta un 62%.

II. DESCRIPCIÓN DEL PROBLEMA U OPORTUNIDAD

La ruta que sigue el transporte de un producto desde su origen hasta su destino final es un problema común y relevante dentro de cualquier industria que preste servicios de despacho, debido a los costos asociados al transporte y a la posibilidad de una disminución de estos en caso de implementarse una ruta óptima que considere las restricciones particulares de cada Empresa.

El caso tratado en este documento corresponde al de una importante Empresa productoras de alimentos. Esta se encarga directamente de la distribución de estos productos a sus clientes, entre los que se encuentran supermercados, restaurantes, mayoristas y pequeños almacenes. Para esto, cuenta con una serie de centros de distribución a lo largo del país, donde cada uno sirve una zona geográfica determinada.

La cantidad de producto demandada por parte de estos clientes presenta una estacionalidad semanal. Esto significa que semana a semana se ve el mismo patrón de comportamiento, donde se reciben menos pedidos al comienzo de la semana y más a medida que se acerca el día viernes. Para los fines del presente trabajo se utilizarán los datos de solo un CD, del año 2016; en estos, la mínima demanda (sin considerar la demanda de días posteriores a festividades) fue de 248 pedidos el 26 de marzo y 516 pedidos la máxima, el 2 de octubre.

Para realizar estos despachos, en cada CD se cuenta con una flota de camiones licitados de 5 toneladas de capacidad, con estos se realiza la distribución a los clientes de la zona asignada. Los pedidos se reciben diariamente y son despachados el día siguiente.

La estacionalidad hace evidente un problema en el despacho de productos, ya que la demanda solo es satisfecha a cabalidad los días de baja demanda, no así en los días en que esta es alta.

Otro dato relevante del problema es que la Empresa no realiza ningún tipo de optimización, ni cuenta con herramientas de soporte para establecer las rutas de reparto, siendo estas determinadas por el operario a cargo. Uno de los criterios utilizados es que los clientes están asignados a un camión específico, siendo servidos casi siempre por el mismo. La justificación de esto es que permite a los choferes conocer las rutas para llegar al cliente y también al cliente mismo, disminuyendo los tiempos de transporte y de servicio.

La situación descrita afecta directamente en los beneficios de la Empresa, ya que la demanda que no es satisfecha en los días de alta demanda alcanza aproximadamente un 30%, lo que significa dejar de entregar 8 toneladas de producto en promedio, en un solo día.

Frente a este problema, existe la oportunidad de mejorar la asignación de porteadores y clientes, así como el establecimiento de rutas de entrega que maximicen la cantidad de pedidos entregados, mejorando el nivel de servicio de la Empresa.

En primer lugar, se aplicará al caso el problema de ruteo de vehículos. Este es un problema básico de investigación de operaciones, en el que se trazan las rutas de una flota de vehículos para satisfacer la demanda de una serie de clientes, teniendo como objetivo minimizar el costo las rutas.

El problema ha sido ampliamente estudiado desde su planteamiento por Dantzig y Ramser en 1959 [1], fecha desde la cual el aumento en la investigación y consiguiente literatura disponible ha expandido el problema desde su formulación básica a problemas más complejos; incorporando nuevas formulaciones, variantes con nuevas restricciones y desarrollo de algoritmos para la resolución de estas mismas.

Una de estas variantes es el problema de ruteo de vehículos consistente (ConVRP) [22], el cual incorpora restricciones para que cada cliente sea servido siempre por un mismo camión. Algunas falencias de este modelo son que no considera ventanas de tiempo —una restricción común en operaciones y problemas de ruteo— y su diseño, que requiere una demanda conocida para programar rutas futuras. Esto hace que no sea aplicable al caso de la Empresa que se trata, ya que ellos conocen la demanda el día anterior.

De todos modos, luego de resolver el problema básico de ruteo, se evaluará el impacto en la consistencia de este al incluir un ConVRP modificado. La modificación incorpora la consistencia en los modelos utilizando incentivos y penalizaciones, de manera que esta aumente sin que esta sea una restricción, como lo es en el modelo original de ConVRP.

Esta evaluación responde a reflejar de mejor manera la intención de la Empresa, que está interesada en ser consistente, más no de forma absoluta ni como prioridad por sobre la disminución de costos.

III. INTRODUCCIÓN

SITUACIÓN DE LA EMPRESA

Ventas y logística externa

La Empresa tiene dos formas de despacho, principalmente el despacho se realiza desde alguno de los CD propios y desde estos a los clientes; además, en el caso de Walmart, se despacha directamente desde la planta de la Empresa a los CD del cliente.

En el caso de la distribución desde los CD, esta se realiza utilizando una flota de camiones ligeros licitados.

En total, la Empresa cuenta con 17 CD distribuidos en tres zonas geográficas (norte, centro y sur) determinadas por la Empresa a lo largo del país.

	Zona Norte	**Zona Centro**	**Zona Sur**
Ciudad en que se encuentra el CD	Arica Iquique Calama Antofagasta Copiapó Coquimbo	Valparaíso Melipilla Santiago	Rancagua Talca Chillan Concepción Los Ángeles Temuco Valdivia Puerto Montt

Tabla 1: Centros de distribución de la Empresa en Chile

Entre los clientes se encuentran locales de centros comerciales, hoteles, restaurantes y casinos, así como distribuidores mayoristas y minoristas.

En la base de datos de la Empresa, la clasificación se resume en siete categorías, expuestas a continuación con los mismos nombres asignados en esta:

1. **Distribuidores:** Distribuidores de alimentos y abarrotes locales.
2. **Grandes Cadenas:** Salas de ventas de grandes cadenas nacionales, corresponden a Alvi Supermercado Mayorista, Cencosud, Tottus, Montserrat, Unimarc y Super 10.
3. **HORECA:** Sigla para referirse a hoteles, restaurantes y casinos; se consideran entre ellos casinos de centros educacionales y de salud, como Sodexo, diversos institutos y Hospitales.

4. **Mayorista:** Comercializadoras mayoristas locales.
5. **Minoristas Chicos:** Pequeños minoristas locales, como minimarkets, almacenes y carnicerías.
6. **Otros:** Empresas no correspondiente clasifican en otras categorías, principalmente transportistas de carga o transporte no regular de pasajeros.
7. **Supermercados:** Supermercados locales, no pertenecientes a grandes cadenas.

En todos los CD el proceso de venta y distribución comienza con la recepción de pedidos, que son recibidos día a día hasta las 19:00 a través de alguno de los siguientes medios:

Web: Carrito de compras web. utilizado tanto por Supermercados y Mayoristas, como por minoristas chicos, HORECA y Otros.

PDA: Vendedores de ruta (cerca de 200) realizan toma pedidos por medio de una aplicación en smartphone, directamente en locales de clientes. Es utilizado frecuentemente en supermercados, mayoristas y distribuidores

Digitado: Vendedores de ruta registran los pedidos tomados a clientes en su estación de trabajo. Mismo uso que medio PDA.

Call center: Centro de llamados en que se reciben pedidos directamente por clientes. Principalmente utilizado por minoristas chicos, HORECA y Otros.

EDI: A través del portal Comercionet -plataforma de intercambio electrónico de datos-, las salas de ventas de grandes cadenas envían órdenes de pedidos, que son recibidas en batchs cada 20 min. Medio utilizado por clientes de grandes cadenas.

De esta forma, la demanda de productos para el día siguiente es conocida.

El problema se presenta con respecto a la demanda futura, ya si bien existen proyecciones de esta en las cuales se basa la distribución desde la planta a los CD, la demanda particular que enfrenta el CD para cada uno de los días siguientes es incierta, considerando que los pedidos serán recibidos siempre el día anterior a su despacho. Esto imposibilita la reorganización de entregas entre días.

Si el stock no es suficiente; el CD Santiago se cuenta con un software que automáticamente asigna el stock disponible, con un algoritmo que distribuye la escasez[1] y reemplaza productos faltantes.

En regiones no se cuenta con este software, por lo que el Jefe de Ventas y el Jefe de Operaciones —Uno por CD — son los encargados de decidir con qué se reemplazará a los productos que faltan, luego de cerrar la recepción de pedidos. Asimismo, es el Jefe de Operaciones el responsable de la planificación de rutas de entrega, también sin apoyo de software.

Para la planificación de las rutas se deben tener en cuenta las ventanas de tiempo en la que cada cliente puede recibir el pedido. En donde cada categoría de cliente cuenta con su propia ventana temporal, y tiempos de servicio aproximados. En estos tiempos de servicio se considera el tiempo de entrega y de espera.

CATEGORÍA	VENTANA DE TIEMPO	TIEMPO DE SERVICIO (H:MM)
DISTRIBUIDOR	8:00 - 12:00	0:30
GRANDES CADENAS	6:30 - 14:00	1:00
HORECA	8:00 - 12:00	0:20
MAYORISTA	8:00 - 18:00	0:30
MINORISTA CHICO	8:00 - 19:00	0:20
OTROS	8:00 - 19:00	0:20
SUPERMERCADO	6:30 - 14:00	0:30

Tabla 2: Ventanas de tiempo y tiempos de servicio por categoría de cliente.

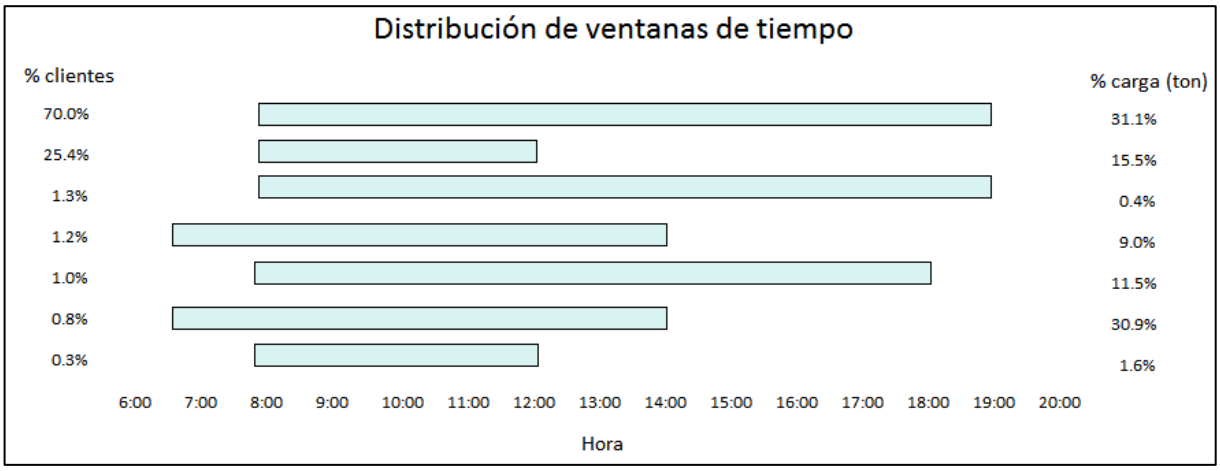

Figura 1: Distribución de ventanas de tiempo

[1] Programa confidencial desarrollado por la Empresa.

Los pedidos a entregar son preparados durante la noche para cargar los camiones (porteadores) a las 4:00 horas, a cargo del Jefe de Despachos, de forma que los camiones puedan salir del CD desde las 5:00; con una jornada laboral hasta las 19:00. En esta jornada los porteadores realizan la totalidad de la ruta, sin volver al CD.

La venta se completa con la facturación del pedido, que se realiza una vez que el porteador entrega los productos, en el lugar de la entrega, por medio PDA.

Algunos pedidos pueden ser no completados, por motivos como cliente fuera de ruta —que no se asignó a ninguna ruta—, pedido falso, cliente sin dinero, local cerrado, reemplazo de productos no aceptado, sector peligroso, condición de fecha de caducidad no cumplida, falla de calidad o cajas dañadas. Estos corresponden a 8% de los pedidos, donde el 5% corresponde a rechazos por parte del cliente.

La flota de camiones consiste en vehículos con capacidad de 5 ton. En general, en una ruta los camiones no completan su capacidad de carga, de forma que esta no es un factor limitante para la operación. En contraste, la hora de regreso al CD sí es un factor limitante, en donde algunos días los camiones finalizan la ruta sin poder entregar todos los pedidos.

L	M	W	J	V	S
14:00	15:00	17:00	19:00	19:00	14:00

Tabla 4: Hora de regreso promedio aproximadas, por día, de porteadores a CD[2].

Distribución interna

La Empresa cuenta con 4 plantas productoras en el país:

- **Planta 1:** Planta principal, produce solo para el mercado nacional.
- **Planta 2:** Principalmente produce para exportación, y menormente para mercado nacional.
- **Planta 3:** Planta para mercado de zona norte grande.
- **Planta 4:** Producción de productos secundarios.

En la Planta 1 se consolidan todos los productos destinados al mercado nacional, a excepción de zona norte grande, enviados desde la Planta 2, Planta 4 y proveedores externos. Desde esta se

[2] Datos aproximados de la Gerencia de Logística, no se cuenta con base de datos sobre esta información.

envían los productos a los CD de la Empresa. Esto se debe a que desde las plantas se envían solo camiones complemente cargados, y los CD no tiene capacidad para la recepción de ese volumen de carga.

La toma de pedidos para reabastecimiento de los CD se decide en coordinación del Jefe de Ventas y Jefe de Operaciones, en base a pronósticos de demanda de productos.

CD Melipilla

En este trabajo se utilizarán los datos de pedidos del año 2016 de la sucursal Melipilla, que abastece clientes de las regiones V, VI y RM. Esta cuenta con 15 camiones de capacitad de carga de 5 ton. En la figura 2 se muestran las comunas servidas por este CD, y en la tabla 4 la cantidad de clientes por comuna, clasificados por categoría. Se considera como cliente un local que realiza pedidos, aun siendo varios locales de la misma Empresa.

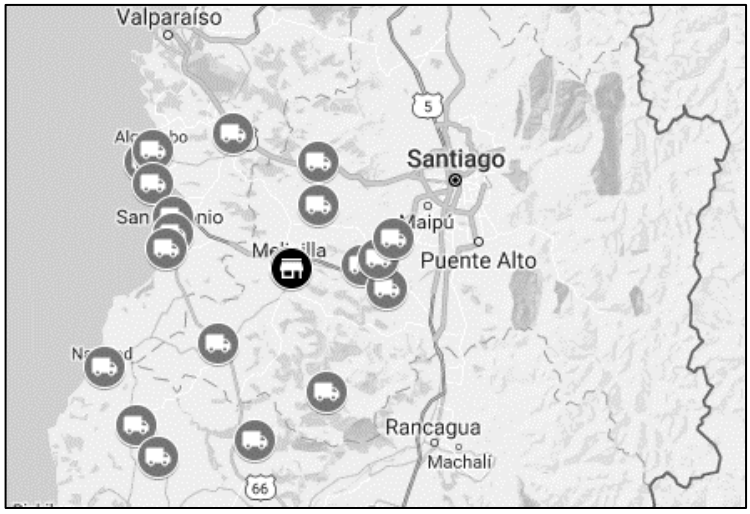

Figura 2: Zona abastecida por CD Melipilla.

	CATEGORIA / COMUNA	DISTRIBRIDOR	GRANDES CADENAS	HORRCA	MAYORISTA	MINORISTAS CHICOS	OTROS	SUPERMCADO	TOTAL
RM	ALHUE			14		43			**57**
RM	CURACAVI		2	23		84	1		**110**
RM	EL MONTE	2	1	30	1	136		3	**173**
RM	ISLA DE MAIPO		2	19	1	121		1	**144**
RM	MARIA PINTO			1		50			**51**
RM	MELIPILLA	4	2	125	10	329	29		**499**
RM	PENAFLOR	1	4	61	4	207	1		**278**
RM	SAN PEDRO			7		27		1	**35**
RM	TALAGANTE		4	70	6	216	3		**299**
	TOTAL	**7**	**15**	**350**	**22**	**1213**	**34**	**5**	**1646**
V	ALGARROBO			39	1	57		4	**101**
V	CARTAGENA		1	111	3	189	2	3	**309**
V	CASABLANCA		2	26		54		1	**83**
V	EL QUISCO	1	1	52	1	98		1	**154**
V	EL TABO	1	1	50	1	79		2	**134**
V	SAN ANTONIO	2	4	119	3	362	3	4	**497**
V	SANTO DOMINGO		1	4		12		5	**22**
	TOTAL	**4**	**10**	**401**	**9**	**850**	**5**	**20**	**1299**
VI	LA ESTRELLA			1		8		1	**10**
VI	LAS CABRAS					3			**3**
VI	LITUECHE			7		17	1	1	**26**
VI	NAVIDAD			17		29		1	**47**
	TOTAL	**0**	**0**	**25**	**0**	**57**	**1**	**3**	**86**
	TOTAL GENERAL	**11**	**25**	**776**	**31**	**2120**	**40**	**28**	**3031**

Tabla 4: Clientes por comuna que realizaron pedidos a CD Melipilla.

Los minoristas chicos son la categoría que concentra la mayor cantidad de clientes (70%) y pedidos (65%), sin embargo, sus pedidos son pequeños, 31 kg en promedio, por lo que su volumen corresponde al 31,1% del total de los pedidos. En contraste la categoría de grandes cadenas representa a menos del 1% de los clientes del CD, y pocos más de un 4% de los pedidos, pero el tamaño de estos pedidos es en promedio 483 kg, por lo que representan un 31% del volumen de los pedidos del CD. De similares características son los grupos de mayoristas y supermercados, que en promedio realizan pedidos de 285 kg y 186 kg respectivamente; tendiendo cada uno casi un 10% del total de los pedidos.

Al analizar la periodicidad de los pedidos de cada una de las categorías, se observa que en general durante el año se mantiene estable el número de pedido por cada una de las categorías.

CATEGORIA	NUM. CLIENTES	% CLIENTES	VOL. PEDIDOS (TON)	%VOL	NUM. PEDIDOS	% PEDIDOS	KILOS/ PEDIDO
DISTRIBUIDORES	10	0.3%	101	1.6%	434	0.4%	232
GRANDES CADENAS	25	0.8%	1986	30.9%	4111	4.2%	483
HORECA	770	25.4%	995	15.5%	22875	23.4%	43
MAYORISTA	29	1.0%	739	11.5%	2596	2.7%	285
MINORISTAS CHICOS	2121	70.0%	2003	31.1%	63879	65.4%	31
OTROS	40	1.3%	27	0.4%	621	0.6%	44
SUPERMERCADO	36	1.2%	582	9.0%	3130	3.2%	186
TOTALES	**3031**		**6433**		**97646**		

Tabla 5: Distribución de clientes, volumen total de pedidos y número de pedidos al CD Melipilla de enero a octubre 2016.

CATEGORIA\MES	ENE	FEB	MAR	ABR	MAY	JUN	JUL	AGO	SEP	OCT
DISTRIBUIDORES	32	34	32	50	50	51	50	50	40	45
GRANDES CADENAS	418	406	443	417	396	418	411	431	367	404
HORECA	2563	2761	2098	2112	2125	2041	2278	2305	2221	2371
MAYORISTA	219	284	272	280	261	289	273	273	210	235
MINORISTAS CHICOS	6802	6623	6519	6313	6016	6025	6498	6649	6126	6308
OTROS	54	70	69	50	56	68	70	71	53	60
SUPERMERCADO	330	358	315	321	302	295	315	320	271	303
TOTAL	**10418**	**10536**	**9748**	**9543**	**9206**	**9187**	**9895**	**10099**	**9288**	**9726**

Tabla 6: Cantidad de pedidos por categoría y mes enero a octubre 2016.

En promedio, un camión de la flota atiende a 27 clientes al día. Como se mencionó anteriormente, la hora de regreso de los camiones es un factor limitante en la operación ciertos días de la semana. Esto se debe al aumento de cantidad de pedidos y del tamaño de los mismos a partir del día miércoles, alcanzando los valores más altos los días jueves y viernes. Los días de mínima demanda fueron el 20 de septiembre y 2 de enero, con 204 y 228 pedidos respectivamente, sin embargo, corresponden a días próximos a festividades. Sin considera este factor, la menor demanda fueron 248 pedidos el 26 de marzo. La mayor cantidad de pedidos se presentó el 2 de octubre, donde se contabilizaron 516 pedidos.

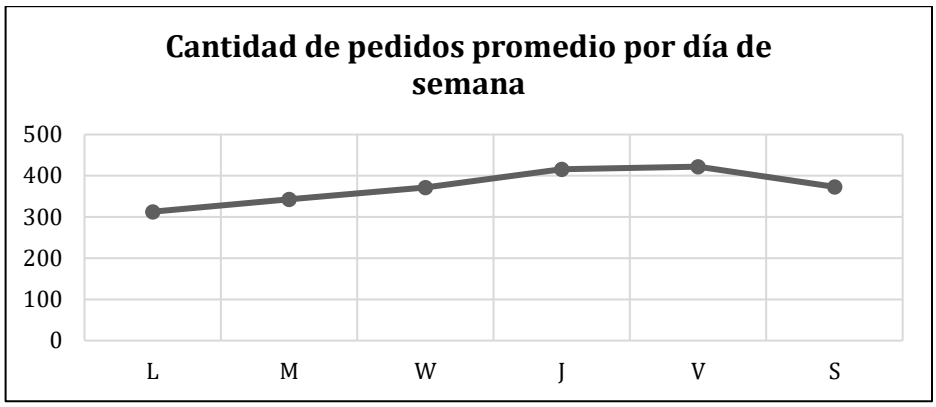

Figura 3: Cantidad de pedidos promedio por día de semana, enero a octubre 2016.

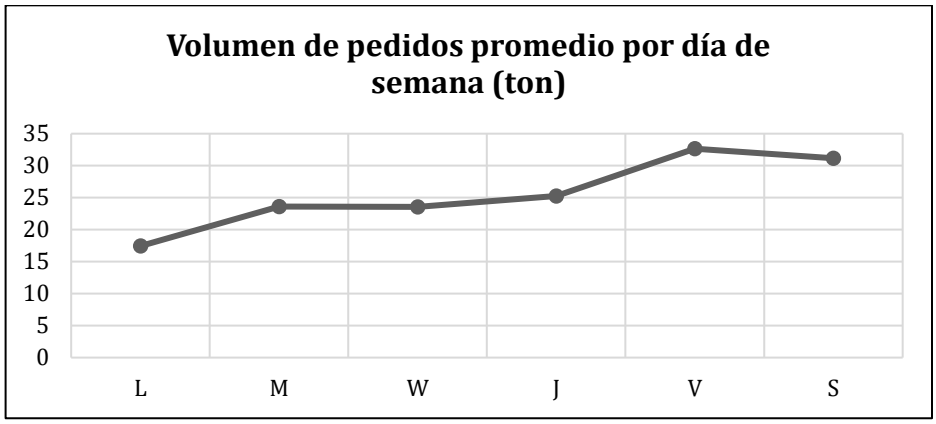

Figura 4: Toneladas de producto pedidas promedio por día de semana, enero a octubre 2016.

Para un análisis más detallado se utilizarán los datos del mes de enero 2016, mes en que se registraron la mayor cantidad de volumen de producto pedidos (714,8 ton). En este mes se tiene un comportamiento similar de cantidad de pedidos por día que al promedio del año completo.

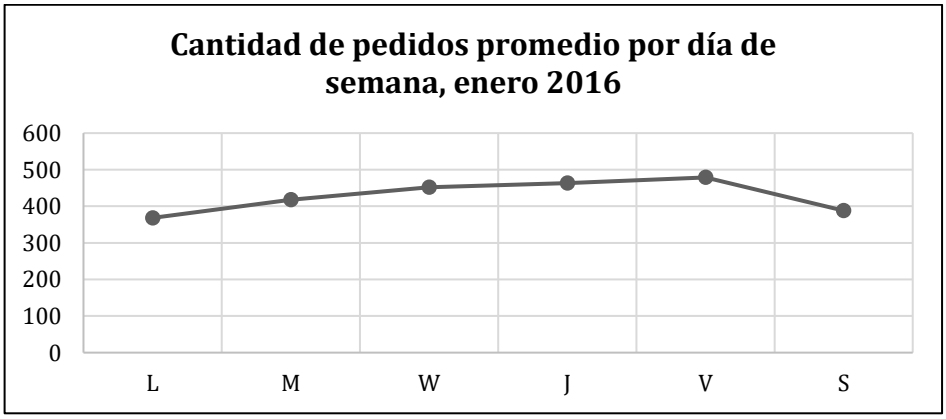

Figura 5: Cantidad de pedidos promedio por día de semana, enero 2016.

En este mes, en promedio a cada camión le fueron asignados 27,7 clientes por día. En la figura 6 se encuentra el histograma de la cantidad de clientes asignados por camión durante todo el mes de enero. Mientras que en la figura 7 se muestra el día de menor demanda en enero, (lunes 18 de enero 2016, total de 342 pedidos; ignorado el 2 de enero con 228). En la figura 8 el día de mayor demanda en el mes (viernes 22 de enero 2016, total de 473 pedidos). Se observa como con la misma cantidad de camiones, el viernes cada camión tiene mayor cantidad de clientes a servir (31,6 en promedio), versus el lunes (22,8 en promedio)

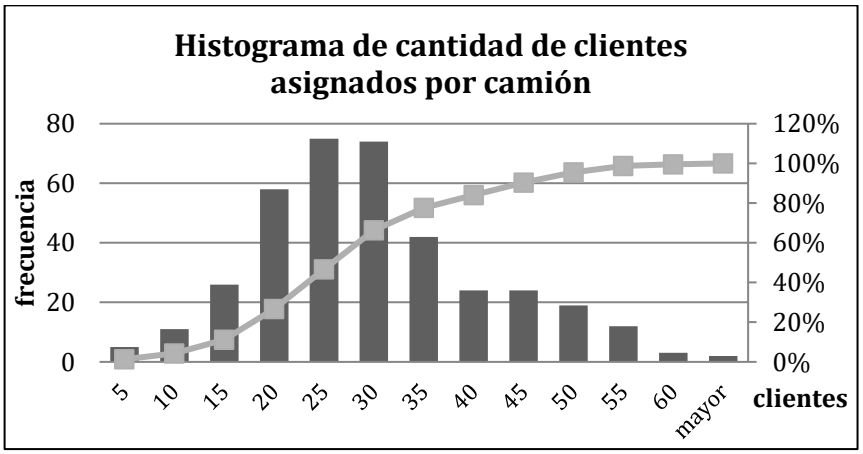

Figura 6: Histograma de cantidad de clientes asignados por camiones, enero 2016.

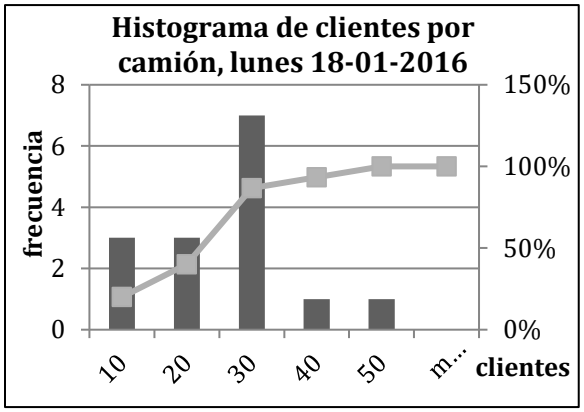

Figura 7: Histograma de clientes asignados a camión, lunes 18-01-2016.

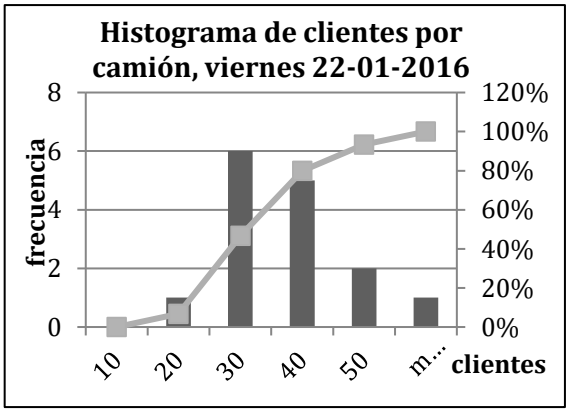

Figura 8: Histograma de clientes asignados por camión, viernes 22-01-2016.

La ocupación (de capacidad de carga, en toneladas) de los camiones en promedio es del 38%. Sin embargo, un 37% de las veces, los camiones tienen una ocupación inferior o igual al 30%, y un 14% menor al 20% de ocupación. Estos casos se deben a pedidos en días de baja demanda, en pequeñas localidades, donde a pesar del pequeño tamaño del pedido, hay más de un porteador en el sector. Situaciones como esta son causadas porque en rutas de días anteriores estos camiones fueron asignados a ese sector, y al no haber problemas de falta de tiempo en días de baja demanda, se mantiene el mismo esquema de ruteo para facilitar la labor de Jefe de Operaciones.

Por otro lado, es menos frecuente que los camiones estén muy cargados. Un 22% de las veces los camiones tienen ocupación superior al 50%, y los un 11% de las veces esta es mayor al 60%. Sin embargo, es en estos casos cuando algunos pedidos dejan de ser entregados, logrando atender a un 71% de los clientes[3].

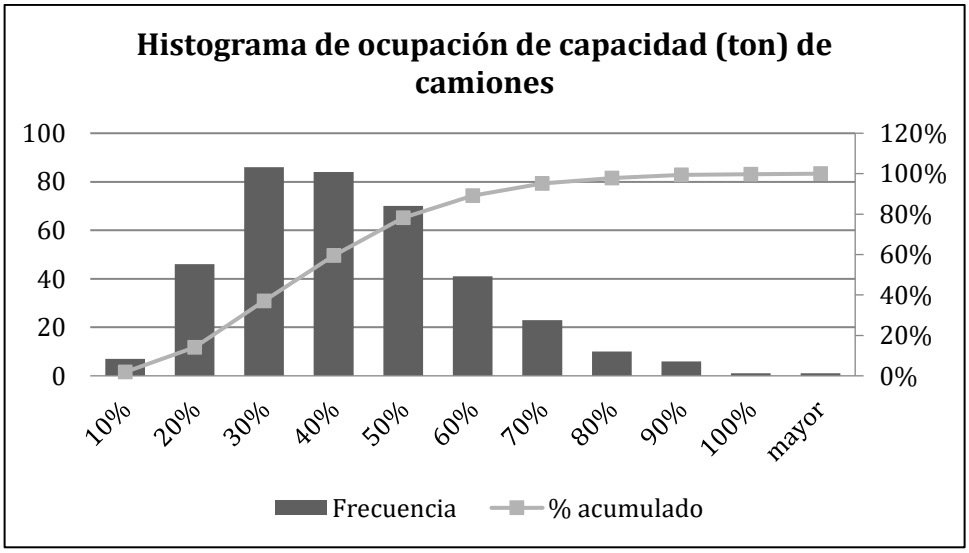

Figura 9: Histograma de ocupación de capacidad de carga de camiones, enero 2016.

[3] Información de la Gerencia de Logística, a la espera de recibir datos sobre ventas.

En la tabla 7 se disponen la cantidad de clientes que atendió cada porteador en cada comuna, en el primer grupo se encuentran las comunas de la Región Metropolitana, en el segundo los de la V Región, y en el tercer los de la VI Región. Se observa que los porteadores tienden a estar agrupados en sectores, o algunos que atienden específicamente una comuna, o grupos de comunas.

PORTEADOR / COMUNA	556	564	565	568	697	833	879	1425	1949	2715	2794	2974	3493	4180	4249	4251	4251
PENAFLOR	39				21	102				1		1		1	1	21	125
TALAGANTE	4			110	32	3			1	122			152	67	6	14	
MELIPILLA		131		3	55	3	2		185	1		3		4			
EL MONTE	51					2								115		5	
CURACAVI	89																
MARIA PINTO	32																
ISLA DE MAIPO					12					98				1	97		
SAN PEDRO	1																
CARTAGENA	1	1	1			99		1			9	12					
SAN ANTONIO			135		31	1	180	142									
CASABLANCA	51											2					
EL QUISCO											47	121					
EL TABO						20					108	71					
SANTO DOMINGO							21	1									
ALGARROBO							1				2	66					
LITUECHE			1													35	
NAVIDAD			1													17	
LA ESTRELLA																7	
LAS CABRAS																3	

Tabla 7. Cantidad De clientes servidos por porteador en comuna en enero 2016.

Por otro lado, se verifica si los porteadores tienen tendencia a servir solo a un tipo de cliente, en la tabla 8 se presenta el resumen de cantidad de clientes por tipo servidos por cada camión del CD Melipilla en enero 2016, sin encontrarse preferencias de este tipo.

PORTEADOR / CATEGORIA	556	564	565	568	697	833	879	1425	1949	2715	2794	2974	3493	4180	4249	4251	4251
DISTRIBUIDORES		1	2	1		1			1		2	3		2			
GRANDES CADENAS	3	1	1	1	4	1	2	2	1	5	2	3	2	2	4	3	3
HORECA	55	33	30	12	40	62	41	40	30	42	51	81	35	32	14	24	24
MAYORISTA	6	6	2	1	3	1	1	3	6	1	1	2	5	1	1	2	2
MINORISTAS CHICOS	196	80	98	95	102	158	152	98	144	168	108	178	102	147	84	72	95
OTROS	1	10	2	1	1	1	2	1	2	4	1	1	3	1	1	1	1
SUPERMERCADO	7	1	3	2	1	6	6		2	2	1	8	5	3			

Tabla 8: Cantidad de clientes por categoría servidos por porteador, enero 2016.

Por último, se revisa la cantidad promedio de porteadores distintos que atienden a un cliente. Se obtiene que cada cliente es atendido por 1.11 porteadores distintos, con una desviación estándar de 0.32.

Conclusiones

Se puede apreciar que la Empresa se encuentra en una situación en la que los pedidos muestran una clara estacionalidad semanal, pasando de demandas promedio de 15 a 20 toneladas de producto la primera mitad de la semana, y alrededor de 300 clientes, a más de 30 toneladas y cerca de 400 clientes para el fin de semana. Siendo las cotas inferior y superior 204 y 516 clientes respectivamente. También se tiene un número limitado de camiones que no puede satisfacer a la totalidad de los clientes los días de alta demanda. Esto explica la necesidad de la Empresa de mejorar el uso de flota de porteadores para estos días. Esto último también se debe a que la demanda no puede ser reorganizada dentro de la semana, ya que los pedidos son tomados diariamente, y la entrega se realiza al día siguiente. Finalmente, existe predisposición a que los porteadores sirvan a los mismos clientes, o zonas específicas.

El problema de los despachos podría resolverse de forma sencilla mediante el uso de metaheurísticas apropiadas para instancias de este tamaño, ya que tiene las características de un problema de ruteo con restricción de capacidad, ventanas de tiempo, tiempos de servicio establecidos y una flota homogénea.

Sin embargo, surge la necesidad de incluir la restricción —o al menos la preferencia— de consistencia en el servicio; a fin de reflejar apropiadamente el modo en que opera la Empresa, y los beneficios prácticos que implica que los mismos camiones tiendan a servir a los mismos clientes.

REVISIÓN DE LITERATURA

Problema de ruteo de vehículos

El VRP es uno de los problemas clásicos de investigación de operaciones. En términos generales, el VRP trata de encontrar la ruta para una flota de vehículos que deben dar servicio a una serie de clientes. Ha sido estudiado durante más de 40 años, desde su introducción por Dantzing y Ramser en 1959 [1], siendo un problema de optimización desafiante, por la gran cantidad de variables en el mismo, y las distintas restricciones que se le pueden imponer [3].

Los problemas de ruteo de vehículos se clasifican como con o sin restricciones [3], considerándose el problema del vendedor viajero (TSP) el problema de ruteo irrestricto más simple. En este se tienen una serie de ciudades (clientes, nodos), que deben ser visitadas por un vendedor viajero (o camión), el cual debe encontrar una ruta para visitarlas todas, minimizando la distancia total recorrida. En la figura 10 se ilustra un ejemplo del TSP, teniendo a la izquierda la ubicación de las ciudades a visitar, y a la derecha la ruta óptima encontrada.

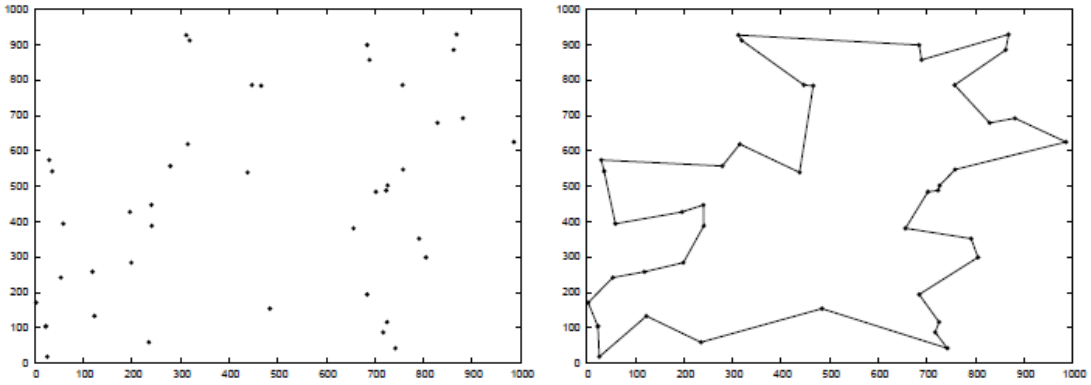

Figura 10: Solución de un TSP. Fuente: Stefan Ropke (2005) [4].

Dantzing y Ramser extendieron el TSP original con su formulación del *problema de despacho de camiones* [1], actualmente considerado como el problema del vendedor viajero múltiple (m-TSP) [19], o por algunos autores simplemente como VRP sin restricciones (refiriéndose a VRP como un problema específico y no a la familia de problemas que representa). Este problema consiste en una versión generalizada del TSP en el cual se tienen m vendedores (o camiones) que deben visitar las ciudades o clientes.

Se tienen dos condiciones:

- Restricción de clientes: Cada cliente debe ser servido por solo un vehículo.
- Restricción de vehículos: Todas las rutas (tours) deben empezar y terminar en el depósito.

Así, en la formulación clásica se tiene un grafo completo $G = (V, A)$, con V definido como un set de vértices $V = \{0\} \cup N$, en donde N representa al set de clientes, y 0 es el depósito. A es el set de arcos, en el cual cada arco (i, j) corresponde a un costo no negativo c_{ij} de ir de un vértice i a uno j. Adicionalmente se tiene un número de vehículos m. Se define una variable binaria x_{ij} como:

$$x_{ij} = \begin{cases} 1 & si \ \ el\ arco\ (i,j)\ pertenece\ a\ la\ solución, con\ i \neq j \\ 0 & en\ otro\ caso \end{cases}$$

$$Min \sum_{i \neq j} c_{ij} x_{ij} \tag{2.1}$$

Sujeto a:

$$\sum_{j \in V} x_{ij} = 1 \qquad \forall i \in V \backslash \{0\} \tag{2.2}$$

$$\sum_{i \in V} x_{ij} = 1 \qquad \forall j \in V \backslash \{0\} \tag{2.3}$$

$$\sum_{i \in N} x_{i0} = m \tag{2.4}$$

$$\sum_{j \in N} x_{0j} = m \tag{2.5}$$

$$\sum_{i \notin S} \sum_{j \in S} x_{ij} \leq |S| - 1 \qquad \forall S \subseteq V \backslash \{0\}, S \neq \emptyset \tag{2.6}$$

$$x_{ij} \in \{0,1\} \qquad \forall i, j \in V \tag{2.7}$$

La función objetivo (2.1) establece la minimización de costos (distancia), de las rutas utilizadas. Las restricciones (2.2) y (2.3) establecen la entrada y salida respectivamente a cada vértice; definiendo que solo un vehículo puede visitar cada cliente. En (2.4) y (2.5) se restringe a que los m vehículos lleguen y salgan del depósito. La expresión (2.6) impone la restricción de conectividad, previniendo la formación de subtours.

En la figura 11 se ilustra un ejemplo de un problema de ruteo, en la izquierda se tiene el set de clientes, y en la derecha las rutas trazadas para los vehículos.

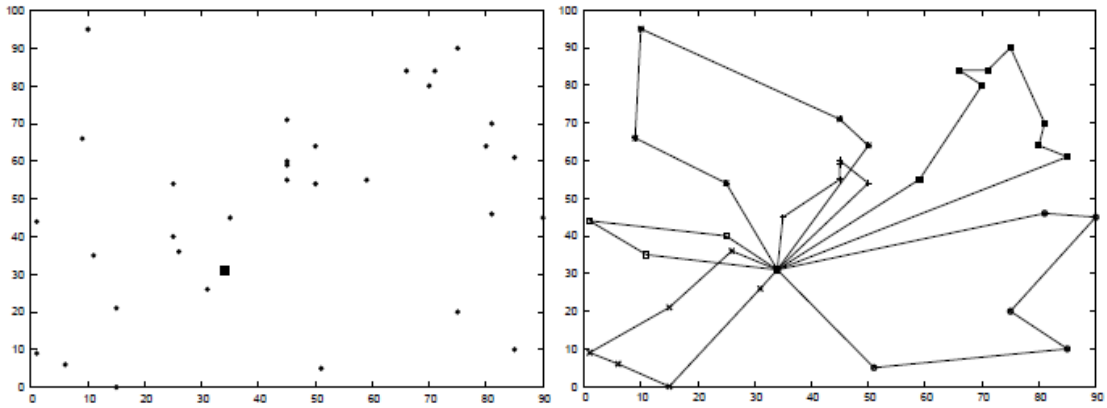

Figura 11: Problema y solución de ruteo. Fuente: Stefan Ropke (2005) [4].

Por otro lado, en el ruteo con restricciones, existen múltiples variantes con las que se ha ampliado la literatura, siendo el CVRP es el más simple y estudiado [3]. Se compone de un set de clientes, de igual forma que el m-TSP, pero cada uno con cierta demanda, que son visitados o servidos por una flota de vehículos con una capacidad de carga definida. Se restringe que la suma de las demandas de los clientes en una ruta (atendida por un vehículo) debe ser menor o igual a la capacidad del vehículo asignado a esa ruta.

Asimismo, otra variante de uso extendido es el VRPTW [2]. Se tiene un problema similar al m-TSP, pero se agrega la restricción de ventanas de tiempo, o sea que los clientes deben ser visitados dentro de cierto horario; además cada cliente tiene un tiempo en que demora ser servido. Algunos autores consideran al VRPTW como una extensión del CVRP, incluyendo la restricción de capacidad, mientras que otros los definen como problemas independientes, siendo el CVRPTW el problema que contiene ambas restricciones.

Métodos de resolución

Para la resolución son utilizados tres tipos de métodos: Algoritmos de solución exacta, Heurísticas y Metaheurísticas [5, 7]. Los algoritmos exactos buscan calcular una solución óptima para el problema. Entre estos métodos se cuentan branch and bound, and bound, branch and cut, generación de columnas, cut and solve, programación dinámica, entre otros [3]. Si bien estos métodos siempre entregan el óptimo del problema, dada la complejidad exponencial del mismo no suelen ser aplicables. Se consideran adecuados para problemas de hasta 50 clientes, aunque recientemente se han podido resolver instancias de tamaño algo mayor a 100; esto depende del solver utilizado y la potencia del equipo.

Debido a las limitaciones de los métodos exactos para la resolución de los problemas de ruteo, el énfasis que los investigadores han dado a este problema se ha concentrado en la resolución mediante heurísticas y metaheurísticas. Estas buscan una buena solución factible, sin garantizar optimalidad.

Las heurísticas se clasifican en heurísticas de construcción y de mejora. Las heurísticas de construcción son aquellas que construyen gradualmente una solución factible, sin contener una fase de mejora. Heurísticas de este tipo son los algoritmos de ahorros, propuestos por Clarke y Wright en 1964 [6], que calcula los ahorros de ir de un cliente a otro, versus ir del mismo cliente a los demás, y en base a esto crea una ruta que contenga la mayor cantidad de ahorros altos; o algoritmos de agrupamiento de dos fases [7], que en la primera fase agrupan a los clientes en clusters, y en la segunda fase se crean rutas para cada uno de ellos.

Las heurísticas de mejora, o de búsqueda local, toman una solución inicial, y la modifican repetidamente con distintos operadores. Si en una iteración se encuentra un nuevo set de rutas con menor costo, este reemplaza a la solución original.

Por otro lado, las metaheurísticas se clasifican en tres grupos, metaheurísticas basadas en búsqueda local, basadas en población, y metaheurísticas híbridas. Se consideran un método poderoso para resolución de problemas combinatoriales, combinando reglas sofisticadas en búsqueda en vecindad, estructura de memoria y recombinación de soluciones. Entre las metaheurísticas basadas en búsqueda local se encuentran los métodos de tabu search (TS) [9, 10, 11, 16], simulated annealing [13] y Greedy randomized adaptive search procedure (GRASP) [12].

Las basadas en población son algoritmos naturales como el algoritmo genético [14] y de optimización basada en colonia de hormigas (ACO) [15]. Finalmente, las metaheurísticas híbridas han sido estudiadas más recientemente, y combinan metaheurísticas y técnicas de optimización [8, 17, 18].

Problema de ruteo de vehículos consistente

Enfocadas en el servicio al cliente, algunas compañías prefieren realizar rutas de despacho en las que los clientes sean servidos en distintos días por los mismos conductores. Esto permitiría profundizar la relación con los clientes, obteniendo beneficios no necesariamente cuantificables, como fidelización y mejor atención para estos, [20], a la vez que mejora el servicio por parte del mismo conductor, al facilitar su labor por tener que servir frecuentemente en los mismos lugares [21]. Estos requerimientos se servicio, juntos cos los del CVRP, definen el ConVRP, en el cual se planifican D días de servicio, para una serie de clientes con demanda mayor o igual a 0 cada uno de los días.

Este problema fue diseñado en su origen para Empresas de paquetería, por lo que no incluye ventanas de tiempo, ya que los clientes residenciales no tienen ventanas de tiempo duras.

Para la resolución del ConVRP se recurre a herramientas similares a las usadas en otros tipos de VRP, pero adicionalmente, se deben incorporar junto a otros algoritmos para resolver las características propias de este problema. De esta forma, existe 2 acercamientos principales para resolverlo.

El primero es el Ruteo a Priori [22], el cual consiste en generar un set de rutas a modo de plantilla para usar todos los días del ruteo. Esta plantilla contiene a todos los clientes que deben ser servidos con una frecuencia mayor a 1, utilizando su demanda promedio; Luego, para cada uno de los días *d* de servicio, se eliminan de las rutas o plantilla los clientes que no deben ser servidos el día *d*, y se agregan a las rutas los clientes que están fuera de la plantilla -con frecuencia de servicio 1- y sí deben ser servidos dicho día. Otro enfoque es el de distritos, o áreas fijas de despacho [23]. Este enfoque se basa en dividir el área de operación en distintas sub-áreas de

despacho, las cuales son atendidas siempre por el mismo vehículo, manteniendo de esta forma la consistencia del servicio. El procedimiento básico para encontrar estas áreas consiste en resolver el problema de ruteo para cada uno de los días de servicio, o una muestra de éstos, y luego calcular cuántas veces un par de clientes aparece en la misma ruta. En base a esto se divide el área del depósito en las K sub-áreas deseadas. Ambos enfoques tienen en común que se encargan de planificar una serie de días seguidos, conociendo la demanda con anticipación, realizando una optimización off-line. Además, en sus formulaciones no se consideran ventanas de tiempo.

IV. HIPÓTESIS, OBJETIVO GENERAL Y OBJETIVOS ESPECÍFICOS DE TRABAJO

a) En el marco del desarrollo del trabajo de título de Ingeniería Civil Industrial, el objetivo principal del trabajo es modelar el problema de despacho desde los CD de la Empresa hasta sus clientes, mediante una variante del VRP, que permita encontrar buenas rutas de entrega para la flota; mejorando de esta manera la utilización de la misma, satisfaciendo la mayor cantidad posible de clientes, y/o maximizando la cantidad de producto entrega.

b) Por otro lado, desde el punto de vista de tesis de Magister de Ciencias de la Ingeniería, se tiene como hipótesis que se puede formular un modelo de ruteo de vehículos, en el que se pueda tener un grado importante de consistencia de servicio, sin tener que planificar una serie de días, y sin que esta sea una restricción dura. Para esto, el principal objetivo es la medición del impacto en la consistencia del servicio en modelos que no la consideren como restricción. De esta forma s encontrará un modelo que entregue buenos resultados desde el punto de vista de la consistencia del servicio.

Acorde a este fin, se desarrollan los siguientes objetivos específicos y la metodología para cumplirlos:

- Revisar el estado del arte: qué se ha desarrollado y cómo se adecúa a las necesidades de la operación; De qué forma esto puede ser utilizado para los objetivos del trabajo.

- Procesar los datos de la Empresa: Se debe recibir información de pedidos, ser procesada para ser posteriormente analizada, así se comprenderán las características de la operación.

- Estudiar los factores y restricciones que influyen en la logística de transporte: Con el objetivo de formular supuestos y restricciones que alimentarán el modelo.

- Determinar método de modelamiento e implementación de un modelo de optimización: De acuerdo con las restricciones y supuestos analizados, se evaluarán formulaciones idóneas y se plantearán alternativas que luego serán evaluadas.

- Validar el modelo con benchmarks y con las instancias disponibles de la Empresa: El modelo deberá ser validado tanto con instancias de prueba de otros trabajos, como con los datos de la Empresa. Se calcularán los beneficios de la operación desde el punto de vista de la Empresa, y se medirá el impacto en la consistencia.

V. METODOLOGÍA

DESCRIPCIÓN DEL MODELO

En función de los objetivos descritos anteriormente, se trabajará con tres variantes de un modelo de ruteo:

1- Ruteo de vehículos con restricción de capacidad y ventanas de tiempo (CVRPTW): Modelo de ruteo que incluye las restricciones de operación de la Empresa sin considerar la consistencia del servicio. Las soluciones entregadas por este modelo serán contrastadas con la información histórica de la Empresa, con el fin de cumplir el objetivo (a). Adicionalmente, estas soluciones serán utilizadas como línea base para definir la pérdida de eficiencia en el ruteo al incentivar la consistencia del servicio con los modelos siguientes; también, se medirá el impacto en la consistencia en el ruteo.

2- CVRPTW con incentivo a aumentar la consistencia a través de aprendizaje de choferes: Se incentivará la consistencia del servicio indirectamente, a través de una disminución de los tiempos de servicio de los clientes cuando son atendidos por los choferes asignados, ya que se considerará que estos los conocen y estos traduce en un servicio más expedito. O sea, que el tiempo de servicio de un cliente *i* al ser atendido por un chofer *k*, dependerá de cuántas veces ha *i* sido visitado antes por *k*. Mientras más visitas, menor es el tiempo de servicio. Por ende, este incentivo es a nivel de los tiempos de ruta, y puede afectar la factibilidad de las soluciones. Así, se define el tiempo de servicio *s* del cliente *i*, atendido por el chofer *j*, que lo ha visitado con anterioridad *x* veces:

$$s(i,k,x) = s_0(i) + K \cdot (1+x)^{-L}$$

Donde $s_0(i)$ es el tiempo de servicio base del cliente *i*, *K* el tiempo de servicio adicional si el chofer *k* no lo conoce, y *L* un parámetro que representa la velocidad del aprendizaje.

3- Modelo CVRPTW con penalizaciones por no respetar consistencia: Se trata de una penalización en la función objetivo, por atender en rutas a clientes que no son conocidos por el chofer de esta.

De esta forma se redefine la función objetivo del VRP, adicionando una penalización si es que el chofer *k* no conoce al cliente *i* al visitarlo:

$$Min \sum_{k} \sum_{i \neq j} c_{ij} x_{ijk} + v_{jk} p_j$$

Donde c_{ij} es el costo del arco del cliente i a j, x_{ijk} una variable binaria con valor 1 si el chofer *k* utiliza el arco ij, v_{jk} es un parámetro binario que indica si el cliente *j* es conocido por el chofer *k* tomando valor 1 y 0 en otro caso, y p_j la penalización por no conocer al cliente *j*.

Se plantea que cada cliente, o categoría de clientes, tenga su propio valor de penalización, para experimentar distintas situaciones, con penalizaciones mayores a ciertas categorías de clientes, de forma de priorizar la consistencia en esta categoría.

Adicionalmente, se experimentará con variantes en las que cada chofer tenga más de un cliente asignado, con el fin de intuir si esto puede ayudar a mejorar la consistencia del servicio, entendiéndola como que cada cliente es visitado por la menor cantidad posible de camiones. Con este mismo fin, también se utilizará una variante del modelo con aprendizaje, que consiste en que, si un chofer visita un cliente no asignado, este pasa a estar en su set de clientes, de forma que se incentivará volver a visitarlo y no visitar otros clientes.

Con respecto al experimento 1, para validar el cumplimiento del objetivo (a) los resultados serán evaluados en base a 6 criterios:

- Clientes en ruta: Cantidad de clientes servidos en las rutas del día.
- Distancia recorrida: Total de kilómetros recorridos en cada instancia.
- Ocupación del camión: Porcentaje de uso de la capacidad de carga del vehículo.
- Demanda entregada: Cantidad de kilos entregados en las rutas.
- Porcentaje de demanda entregada: Porcentaje de kilos de producto entregados, del total de pedidos.
- Nivel de servicio: Porcentaje de clientes servidos, del total de clientes que realizaron pedidos.

Como se mencionó en la situación de la Empresa, la flota de camiones disponibles no siempre es capaz de entregar todos los pedidos. Por esto, es posible que en los resultados del modelo 1 se tenga que para servir a todos los clientes de un día se necesiten más vehículos de los disponibles en la Empresa. Para resolver esto, y poder contrastar efectivamente la situación optimizada con la base se plantea el uso de una situación optimizada restringida. En ésta se seleccionarán las *k* rutas con mayor cantidad de clientes, para compararlas con las *k* rutas de situación base de la Empresa.

Para calcular la situación base, en cada día o instancia se resolverá un TSP para cada ruta de la empresa. Estas rutas consisten en los clientes que tiene asignado cada vehículo, asumiendo que la Empresa realiza la mejor ruta posible con las asignaciones que ya tiene establecidas. En el caso de que una ruta contemple más clientes de los que es posible servir, por las ventanas de tiempo de servicio, solo se tomarán en cuenta los primeros clientes en los que la ruta sea factible.

Por otro lado, para evaluar el impacto en la consistencia del modelo 1, se debe calcular la consistencia de éste. Ya que el CVRPTW se resolverá independientemente para cada día, no existirá necesariamente una relación entre la ruta *k* de un día la ruta *k* del siguiente, por lo que no se puede evaluar directamente si un cliente fue atendido en las mismas rutas o no. Para resolver esto se modelará la asignación de rutas de un día con las del día siguiente como un Problema de Asignación Lineal, en el cual se maximizará la cantidad de clientes en común de las rutas de un día con su ruta correspondiente del día siguiente. Así se obtendrá como resultado qué ruta de cada día fue la recorrida por cada vehículo, lo que permitirá calcular con cuántos vehículos diferentes fue servido cada cliente.

Para los modelos 2 y 3 se requiere una base de conocimiento de choferes a clientes, esta estará dada en primer lugar por la asignación actual de camiones a clientes de la Empresa, de forma que un cliente conocido por un chofer es aquel al que este está asignado previamente. Adicionalmente, se experimentará con otras bases de conocimiento. Estas estarán dadas por la frecuencia de visitas de los camiones a los clientes durante el año, basándose en las instancias resueltas del VRP con el modelo 1, de forma similar a la utilizada por Christofides [23].

De esta forma, se obtiene una matriz de visitas basadas en dichas soluciones, en la cual se tiene cuántas veces fue visitado cada cliente por cada camión. Con esto se realizan las asignaciones cliente-camión resolviendo el problema de asignación máxima, donde cada camión tendrá *n/k* clientes (*n* es el total de clientes y *k* el de camiones). El fin es que a cada camión le sean asignados los *n/k* clientes que más veces visitó.

Se esperaba poder evaluar el impacto de asignar clientes a choferes en base a instancias del 2016 con los datos del 2017, pero estos últimos datos no fueron enviados por la Empresa.

IMPLEMENTACIÓN

Para resolver los modelos planteados, se implementa una metaheurística de optimización, método de resolución más adecuado para el problema, dado el tamaño del mismo, que imposibilita la utilización de métodos exactos.

La metaheurística utilizada será simulated annealing, la cual ha probado ser un método eficiente para la resolución del VRP [3], además de ser algoritmo simple conceptualmente, lo que facilitará su implementación y posterior mantención en la Empresa.

```
Input: f: Función objetivo a minimizar.
Data: Pnew: Nueva solución generada.
Data: Pcur: Solución investigada actualmente en el espacio problema.
Data: T: Temperatura en el sistema, decreciente.
Data: t: Índice de la actual iteración.
Data: ΔE: La energía (valor objetivo), diferencia entre f(Pcur) y f(Pnew)
Output: x: Mejor solución encontrada.

Comienzo
    Pcur <- Solución inicial.
    x <- Pcur
    t <- 1
    mientras ¬criterioDeTermino() hacer:
        T <- getTemperature(t)
        Pnew <- mutate(Pcur)
        ΔE <- f(Pnew)-f(Pcur)
        Si ΔE ≤ 0 entonces:
            Pcur <- Pnew
            Si f(Pcur) < f(x) entonces: x <- Pcur
        Sino:
            Si randomUnif[0,1) < e^(-ΔE/T) entonces: Pcur <- Pnew
        t <- t+1
Fin: retornar x
```

getTemperature(t): Función de cambio de temperatura = $\begin{cases} Ts & si\ t < e \\ Ts/\ln(t) & en\ otro\ caso \end{cases}$

mutate(Pcur): Mutación de solución actual: Búsqueda en vecindad.

Algoritmo 1: Simulated Annealig.

La implementación del algoritmo anterior se encuentra en el anexo 1, por otro lado, la función de mutación mutate(Pcur) se encuentra en el anexo 2. Ésta corresponde a una búsqueda en vecindad, definida como la mejor modificación posible a un set de rutas, utilizando operador de

búsqueda local [24]. Los operadores utilizados son Relocate, Exchange, Two-opt y Two-opt*, los tres primeros corresponden a operadores ampliamente utilizados y de demostrada eficacia, mientras el último es un operador diseñado especialmente para problemas con ventanas de tiempo, ya que intercambia nodos de rutas manteniendo el orden de visitas y, por ende, respetando las ventanas de tiempo [24], estos se ilustran en la figura 12.

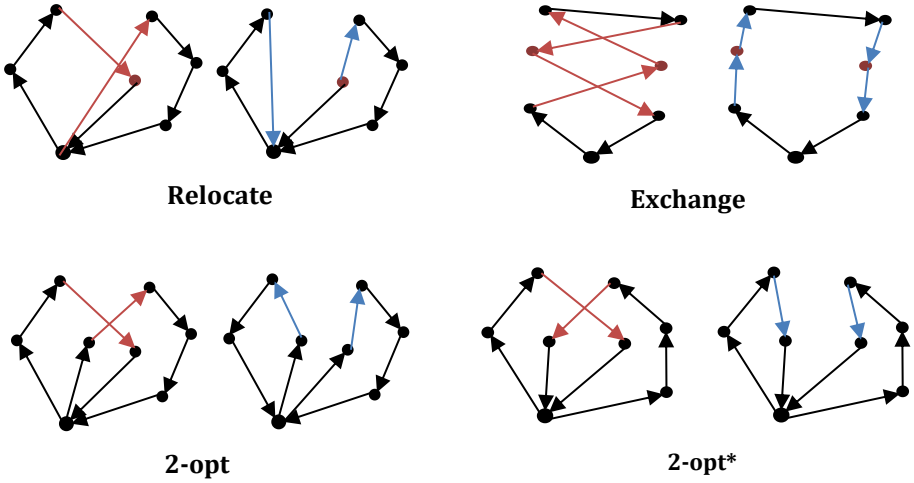

Figura 12: operadores de búsqueda local.

DESCRIPCIÓN DE LOS EXPERIMENTOS

Con estos tres modelos, se resolverán las instancias disponibles de la Empresa. Los resultados del modelo 1 serán analizados para inferir la mejora de la situación base de la Empresa, frente a una situación optimizada. En el caso de que las soluciones de algunos días tengan una cantidad de rutas mayor al número de *n* camiones disponibles de la Empresa, se optará por las *n* rutas con mayor cantidad de producto. Con estas soluciones se contrastará el nivel de servicio, la cantidad de producto entregada, y los kilómetros totales de recorrido de la flota.

Adicionalmente, para estas soluciones se calculará el nivel de consistencia, definido como la cantidad de choferes distintos con los que es atendido cada cliente. De esta forma se cuantificará el impacto en la consistencia del servicio al realizar el ruteo sin esta restricción, vale decir, qué tanto empeora la consistencia del servicio al no considerarla, o qué nivel de consistencia se puede alcanzar sin ser una restricción del modelo.

Por otro lado, las soluciones del modelo 1 será la línea base para la evaluación de los siguientes modelos 2 y 3 en cuanto a optimalidad; calculando cuánto empeora la solución de cada instancia al incorporar consistencia.

Con los modelos 2 y 3 se medirá la consistencia en diferentes experimentos, intentando aumentarla a través de las distintas técnicas descritas anteriormente, tanto de forma general para todos los clientes, como para categorías específicas.

En la tabla 9 se resumen los experimentos realizados, con las características específicas de cada uno.

Experi-mento	Modelo usado	Conocimiento base	Aprendizaje	Penalización
1	1	No utiliza	No utiliza	No utiliza
2.1	2	Rutas Empresa	No	$k = s_0$
2.2	2	Rutas Empresa	No	$k = 2 \cdot s_0$
3.1	3	Rutas Empresa	No	Proporcional a tiempos de servicio
3.1.1	3	Rutas Empresa	Actualización con visitas	Proporcional a tiempos de servicio
3.2	3	Rutas Empresa	No	Igual a todos los clientes
3.2.1	3	Rutas Empresa	No	Igual a todos los clientes
3.3	3	Rutas Empresa	No	Igual a todos y por factor 10 a grandes cadenas
3.3.1	3	Rutas Empresa	Actualización con visitas	Igual a todos y por factor 10 a grandes cadenas
4.1	3	Basado en experimento 1, con 1 camión por cliente	No	Igual a todos los clientes
4.1.1	3	Basado en experimento 1, con 1 camión por cliente	Actualización con visitas	Igual a todos los clientes
4.2	3	Basado en experimento 1, con 2 camiones por cliente	No	Igual a todos los clientes
4.2.1	3	Basado en experimento 1, con 2 camiones por cliente	Actualización con visitas	Igual a todos los clientes
4.3	3	Basado en experimento 1, con 3 camiones por cliente	No	Igual a todos los clientes
4.4	3	Basado en experimento 1, con 4 camiones por cliente	No	Igual a todos los clientes
5.1	3	Basado en experimento 1, con 1 camión por cliente	No	Igual a todos y por factor 10 a grandes cadenas
5.1.1	3	Basado en experimento 1, con 1 camión por cliente	Actualización con visitas	Igual a todos y por factor 10 a grandes cadenas
5.2	3	Basado en experimento 1, con 2 camiones por cliente	No	Igual a todos y por factor 10 a grandes cadenas
5.2.1	3	Basado en experimento 1, con 2 camiones por cliente	Actualización con visitas	Igual a todos y por factor 10 a grandes cadenas
5.3	3	Basado en experimento 1, con 3 camiones por cliente	No	Igual a todos y por factor 10 a grandes cadenas
5.4	3	Basado en experimento 1, con 4 camiones por cliente	No	Igual a todos y por factor 10 a grandes cadenas
6*[4]	3	Rutas Empresa	No	Igual a todos los clientes

Tabla 9: Descripción de los experimentos realizados.

*[4] Modelo sin considerar ventanas de tiempo, para estudiar la influencia de esta en la consistencia del servicio.

VI. RESULTADOS OBTENIDOS

VALIDACIÓN DEL ALGORITMO

El algoritmo implementado fue utilizado para resolver las instancias de benchmark de Solomon, de 100 clientes, y de Gehring y Homberger, de 400 clientes. Con estas instancias se valida la efectividad del algoritmo al converger a la mejor solución conocida de cada problema, y el tiempo que toma en encontrarla. En la figura 13 se muestra el resultado de una de las instancias de 100 clientes, y en la figura 14, una instancia de 400 clientes.

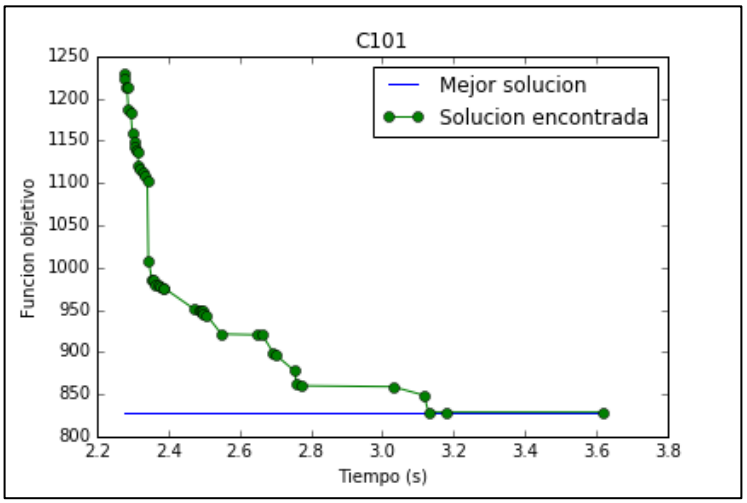

Figura 13: Convergencia de instancia C101 de Solomon.

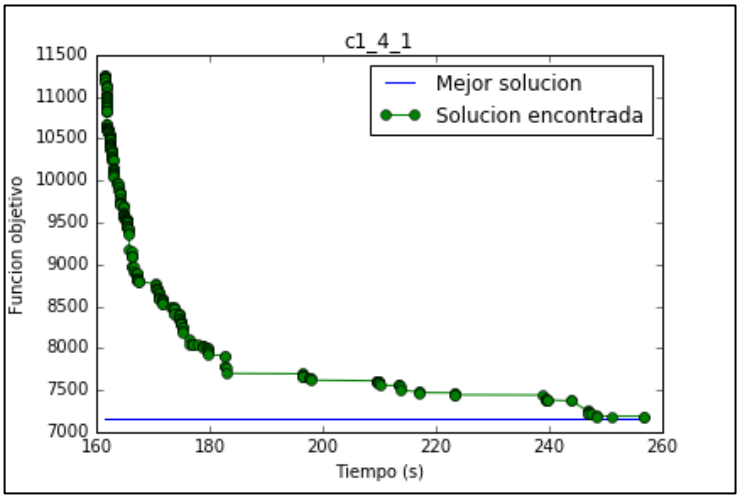

Figura 14: Convergencia de instancia C1_4_1 de Gehring y Homberger.

RUTEO SIN CONSISTENCIA

Para los resultados del ruteo con restricción de capacidad y ventanas de tiempo, sin consideraciones de consistencia, se contrastan los resultados de la optimización en un día de alta y baja demanda de la misma semana, en las tablas 10 y 11 respectivamente.

Baja demanda	Cantidad de rutas	Clientes en ruta	[Km]	Ocupación camión	Demanda entregada [Kg]	% Demanda entregada	Nivel de servicio
Optimizada	12	342	1446	35%	21297	100%	100%
Situación base	14	338	2580	28%	21217	100%	99%

Tabla 10: Resultados en día de baja demanda.

Alta demanda	Cantidad de rutas	Clientes en ruta	[Km]	Ocupación camión	Demanda entregada [Kg]	% Demanda entregada	Nivel de servicio
Optimizada	19	474	2428	41%	39163	100%	100%
Restringida	14	372	1947	52%	36495	93%	78%
Situación base	14	294	2982	41%	28460	73%	62%

Tabla 11: Resultados en día de alta demanda.

De esta forma, en los días de baja demanda se tiene:

- Disminución de 44% de distancia recorrida.
- Aumento marginal de nivel de servicio.
- Disminución de tamaño de flota requerida en 2 camiones.

Por otro lado, en los días de alta demanda:

- Disminución de 35% de distancia recorrida.
- Aumento en 26% de nivel de servicio.
- Aumento en 27% de kg de producto entregados.

CONSISTENCIA

Para cada uno de los experimentos propuestos, se calcula el nivel de consistencia, como proporción de clientes que son servidos por tantos camiones distintos. En la tabla 12 se muestran los resultados, hasta 10 camiones diferentes.

Camiones por cliente\ Experimento	=1	≤2	≤3	≤4	≤5	≤6	≤7	≤8	≤9	≤10
1	0.12	0.27	0.44	0.59	0.72	0.82	0.88	0.92	0.94	0.95
2.1	0.13	0.22	0.32	0.43	0.55	0.65	0.74	0.8	0.85	0.89
2.2	0.12	0.23	0.35	0.48	0.58	0.68	0.75	0.82	0.85	0.88
3.1	0.36	0.59	0.73	0.82	0.88	0.91	0.94	0.96	0.97	0.98
3.1.1	0.37	0.62	0.77	0.86	0.91	0.94	0.96	0.98	0.98	0.99
3.2	0.43	0.65	0.79	0.86	0.90	0.94	0.95	0.97	0.98	0.99
3.2.1	0.49	0.75	0.88	0.94	0.96	0.98	0.99	0.99	1.00	1.00
3.3	0.48	0.70	0.81	0.87	0.91	0.94	0.96	0.98	0.98	0.99
3.3.1	0.58	0.81	0.89	0.93	0.96	0.98	0.98	0.99	0.99	1.00
4.1	0.56	0.75	0.85	0.90	0.93	0.96	0.97	0.98	0.99	0.99
4.1.1	0.62	0.79	0.88	0.93	0.95	0.97	0.98	0.99	0.99	0.99
4.2	0.21	0.67	0.86	0.92	0.95	0.97	0.98	0.98	0.99	0.99
4.2.1	0.19	0.58	0.80	0.90	0.94	0.96	0.98	0.99	0.99	0.99
4.3	0.15	0.34	0.62	0.84	0.92	0.95	0.97	0.98	0.99	0.99
4.4	0.14	0.31	0.53	0.77	0.88	0.94	0.97	0.98	0.99	0.99
5.1	0.59	0.76	0.85	0.92	0.94	0.96	0.97	0.98	0.99	0.99
5.1.1	0.62	0.81	0.89	0.94	0.96	0.98	0.98	0.99	0.99	0.99
5.2	0.21	0.76	0.87	0.93	0.96	0.97	0.98	0.99	0.99	0.99
5.2.1	0.22	0.63	0.81	0.90	0.94	0.96	0.97	0.98	0.99	0.99
5.3	0.16	0.38	0.69	0.86	0.93	0.96	0.97	0.98	0.99	0.99
5.4	0.13	0.29	0.53	0.75	0.87	0.93	0.96	0.98	0.98	0.99
6	0.64	0.82	0.9	0.93	0.95	0.97	0.97	0.98	0.99	0.99

Tabla 12: Proporción de clientes servidos por X camiones por experimento.

A continuación, se comparan los casos de mayor contraste en que se utilizaron las asignaciones de camiones-clientes de la Empresa, y no se modificó la factibilidad de las soluciones. Vale decir la línea base (experimento 1) y el experimento 3.3.1. Se presenta el histograma de cantidad de clientes que son atendidos por tantos camiones, y el acumulativo, en las figuras 15 y 16.

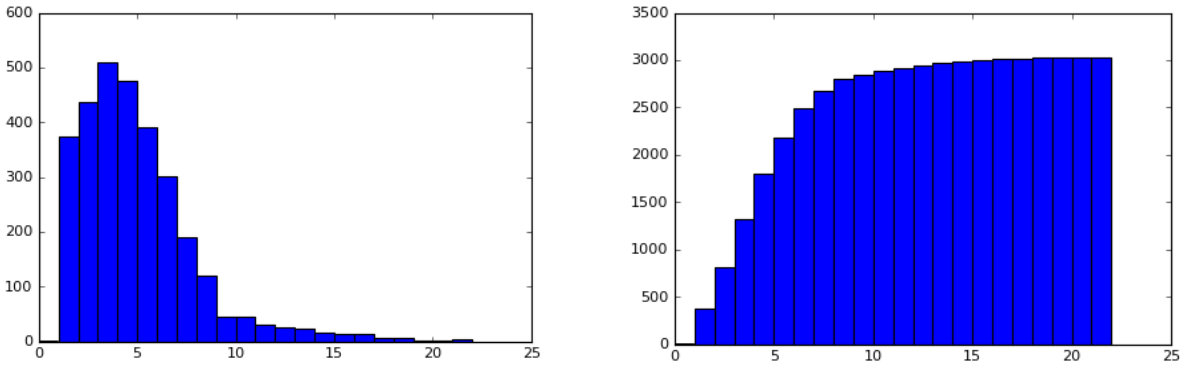

Figura 15: Consistencia de servicio de situación optimizada sin incentivo a consistencia (1).

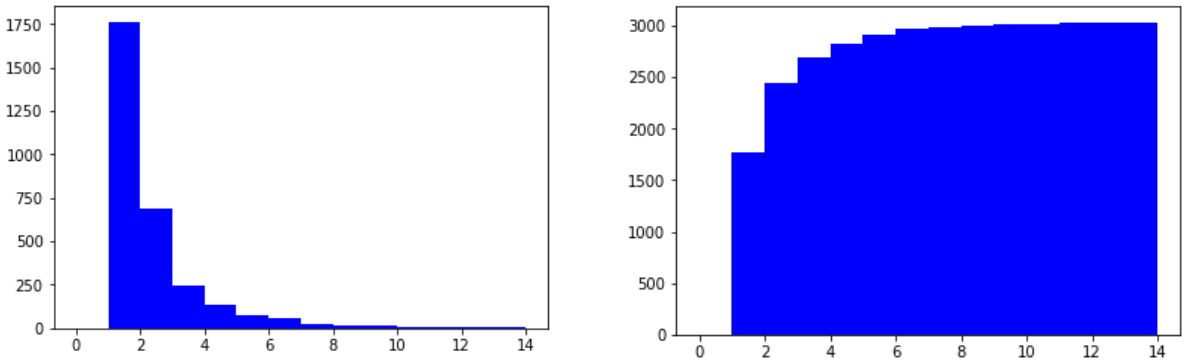

Figura 16: Consistencia en experimento 3.3.1.

En los casos con asignaciones basadas en la frecuencia de visitas de los camiones a los clientes durante el año, se tiene un aumento importante en la consistencia. En la figura 17 se muestran los histogramas correspondientes al experimento 4.1.1.

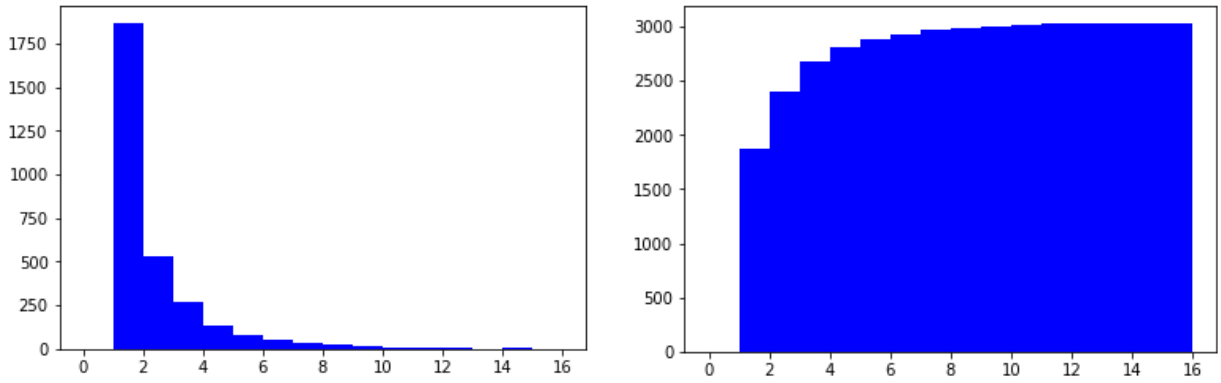

Figura 17: Consistencia en experimento 4.1.1.

Consistencia en categoría Grandes Cadenas

Se calcula el nivel de consistencia para la categoría de Grandes Cadenas, al ser ésta la con clientes con mayor volumen de pedidos y a la vez la con menor cantidad de clientes. Por otro lado, dado sus altos tiempos de servicio se plantea la importancia de mantener la consistencia en esta categoría. En la tabla 13 se presentan los resultados de los experimentos con respecto a la consistencia para dicha categoría.

Camiones por cliente\ Experimento	=1	≤2	≤3	≤4	≤5	≤6	≤7	≤8	≤9	≤10
1	0	0	0	0.08	0.16	0.32	0.56	0.84	0.92	0.96
3.1	0.12	0.12	0.20	0.28	0.40	0.44	0.60	0.76	0.92	0.96
3.1.1	0.08	0.12	0.2	0.32	0.44	0.56	0.72	0.84	0.84	0.92
3.2	0.16	0.20	0.24	0.28	0.40	0.64	0.64	0.76	0.92	0.96
3.2.1	0.12	0.20	0.28	0.44	0.52	0.72	0.88	0.96	0.96	0.96
3.3	0.12	0.16	0.24	0.36	0.44	0.60	0.68	0.76	0.92	0.92
3.3.1	0.24	0.48	0.64	0.72	0.88	0.96	0.96	0.96	0.96	0.96
4.1	0.32	0.48	0.60	0.68	0.76	0.92	0.96	0.96	1.00	1.00
4.1.1	0.40	0.52	0.80	0.92	0.92	0.92	0.92	0.96	1.00	1.00
4.2	0.00	0.56	0.76	0.84	0.88	0.96	0.96	1.00	1.00	1.00
4.2.1	0.00	0.04	0.24	0.32	0.32	0.56	0.68	0.88	0.96	0.96
4.3	0.00	0.04	0.08	0.52	0.80	0.84	0.84	0.96	1.00	1.00
4.4	0.00	0.04	0.16	0.36	0.72	0.88	0.92	0.96	0.96	1.00
5.1	0.28	0.44	0.60	0.76	0.92	0.96	0.96	1.00	1.00	1.00
5.1.1	0.36	0.60	0.80	0.88	0.92	1.00	1.00	1.00	1.00	1.00
5.2	0.00	0.60	0.80	0.88	0.88	0.92	1.00	1.00	1.00	1.00
5.2.1	0.04	0.28	0.56	0.88	0.88	0.88	0.96	0.96	0.96	0.96
5.3	0.00	0.16	0.52	0.72	0.84	0.92	0.96	0.96	0.96	1.00
5.4	0.00	0.04	0.20	0.40	0.56	0.80	0.92	0.96	0.96	1.00
6	0.08	0.36	0.48	0.56	0.68	0.72	0.72	0.76	0.8	0.8

Tabla 13: Proporción de clientes de grandes cadenas servidos por X camiones por experimento.

En la figura 18 se muestran los histogramas para la línea base, en la figura 19 los histogramas del experimento 3.3.1. Ambos experimentos se realizan con la asignación base de la Empresa, siendo los de mayor contraste para este caso.

Por otro lado, en la figura 20 se muestran los histogramas para el experimento 5.1.1, que obtuvo la mejor consistencia para la categoría, utilizando las asignaciones dadas por la frecuencia de visitas durante el año, en particular la con un camión por cliente, pero con aprendizaje.

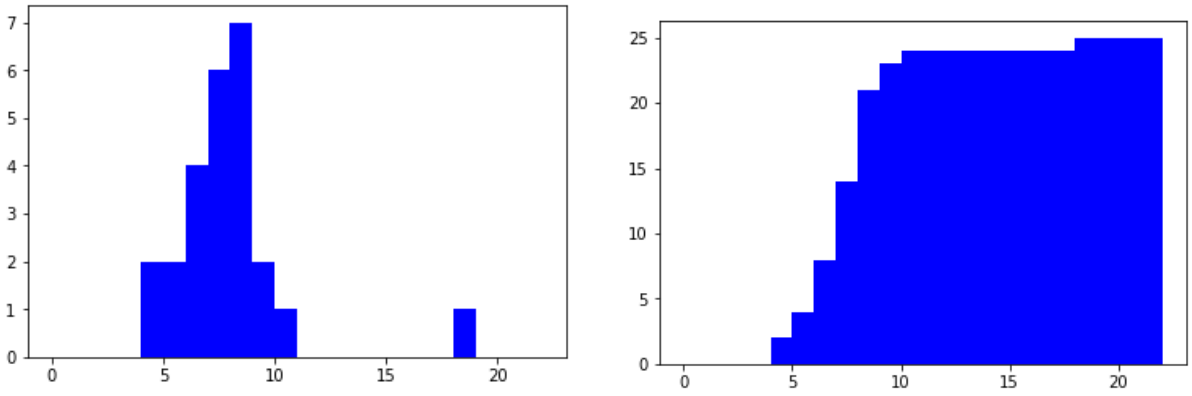

Figura 18: Consistencia para grandes cadenas en situación optimizada sin incentivo a consistencia (1).

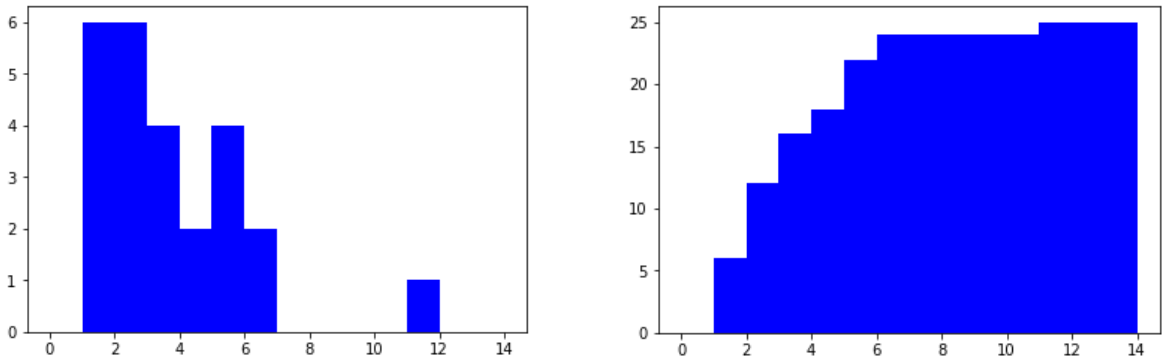

Figura 19: Consistencia para grandes cadenas en experimento 3.3.1.

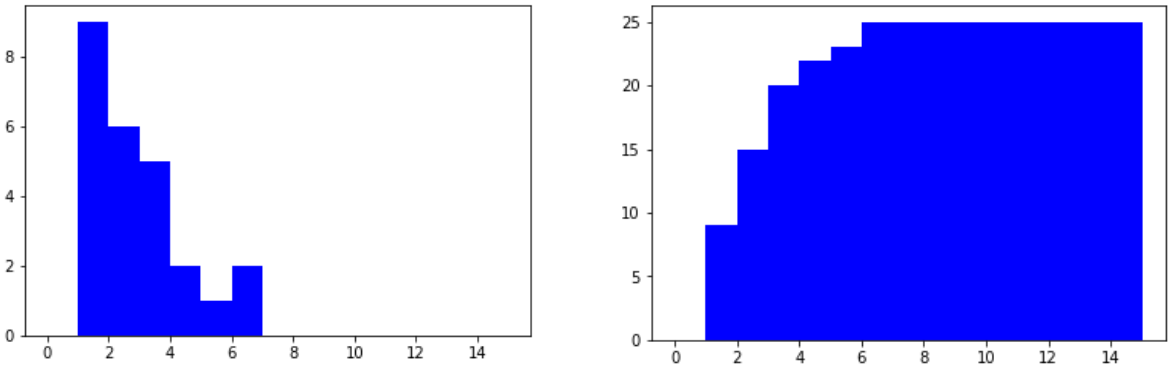

Figura 20: Consistencia para grandes cadenas en experimento 5.1.1.

Impacto de la consistencia en la función objetivo

Para cada una de las instancias disponibles, se calcula el porcentaje de diferencia entre el costo de la línea base (experimento 1) y la solución entregada por los demás experimentos. En la tabla 14 se muestra el promedio de estas diferencias para cada experimento, junto con la varianza de estas.

Experimento	% de Empeoramiento	Varianza
3.1	15%	0.0029
3.1.1	9%	0.0035
3.2	16%	0.0040
3.2.1	11%	0.0054
3.3	23%	0.0076
3.3.1	15%	0.0055
4.1	7%	0.0100
4.1.1	2%	0.0074
4.2	0%	0.0046
4.2.1	8%	0.0039
4.3	-2%	0.0046
4.4	-2%	0.0051
5.1	7%	0.0087
5.1.1	2%	0.0073
5.2	0%	0.0053
5.2.1	-3%	0.0045
5.3	-2%	0.0054
5.4	-3%	0.0050
6	4%	0.0136

Tabla 14: Porcentaje de empeoramiento promedio de soluciones contra experimento 1.

VII. DISCUSIÓN

Sobre el algoritmo utilizado para la resolución del problema, se observa que es capaz de resolver instancias de gran tamaño, tales como las del benchmark de Gehring y Homberger de 400 clientes, y las mismas instancias de la Empresa en tiempos adecuados y similares a los presentados en el estado del arte, siendo estos del orden de los 6 minutos, el que también es manejable a través del número de iteraciones definidas para el algoritmo, pudiendo encontrar buenas soluciones en 3 o 4 minutos; tiempo suficiente para su uso en la Empresa, ya que este solo debe utilizarse una vez al día, al momento de planificar las rutas de entrega.

Por otro lado, las soluciones encontradas por el algoritmo son de calidad aceptable, acercándose a las mejores soluciones encontradas en los problemas de benchmark de gran tamaño, y encontrando la mejor solución conocida en los problemas más pequeños. Esto indica que el algoritmo aún es mejorable, por lo que es recomendable a futuro establecer reglas más sofisticadas para la búsqueda en vecindad.

Aun así, se observa que es capaz de encontrar buenas soluciones para las instancias de la Empresa, en general, disminuyendo la distancia recorrida en las rutas en un 40% aproximadamente. Además, se cumple el objetivo propuesto de mejorar el uso de la flota y aumentar la cantidad de producto entregado, siendo esto más evidente en los días de alta demanda, en los que se aumenta en un 26% el nivel de servicio, y en un 27% los kilos de producto entregados, con respecto a la situación actual de la Empresa.

Con respecto a la consistencia del servicio, se observa que resolver el problema de la Empresa con un modelo de CVRPTW directamente, la empeora considerablemente. Si con la asignación directa de la Empresa se tiene que todos los clientes tienen un solo camión que les realiza las entregas, al optimizar la situación con este modelo, esta consistencia solo se tiene en el 12% de los clientes. Si se analiza cuántos clientes son atendidos por un número razonable de camiones diferentes, el 44% de ellos son atendidos por 3 camiones distintos al realizar el ruteo durante todo el año; y si se amplía este horizonte a 5 camiones diferentes, aún se tiene un 28% de los clientes que son atendidos por más de 5 camiones distintos.

La situación mencionada anteriormente es aún más acentuada al analizar este comportamiento en la categoría de Grandes Cadenas, que se caracteriza por tener una ventana de tiempo acotada, y largos tiempos de servicio. En esta categoría solo se tiene que un 16% de los clientes son atendidos por no más de 5 camiones diferentes. Esto motiva a experimentar con otros modelos que aumenten la consistencia, y también que la prioricen especialmente en esta categoría.

De esta forma, se analizan en primer lugar los experimentos realizados con el modelo 2, el cual trata de incentivar la consistencia a través del conocimiento de los camiones (o choferes) a los clientes. Si un camión conoce a un cliente el tiempo de servicio será menor, y por ende tendrá un incentivo a servirlo. Esta hipótesis no se ve respaldada en los resultados, resultando aún peor la consistencia que en el modelo 1. Esto probablemente se debe a que este planteamiento también modifica la factibilidad de las rutas, ya que en muchos clientes se ve aumentado el tiempo de servicio al ser atendidos por camiones que no los conocen.

Con el modelo 3 se obtuvieron mejores resultados, Incorporando solo una penalización en la función objetivo, y aun sin modificar la asignación de clientes a camiones de la Empresa, se aumentó considerablemente el grado de consistencia del servicio. Con modelos sin aprendizaje, se logró que hasta un 48% (3.3) de los clientes fueran atendidos por un solo camión, y que hasta el 91% por no más de 5 camiones distintos (experimento 3.3). Esta consistencia aumenta a 58% y 96% respectivamente si se considera además el aprendizaje de los choferes al realizar una nueva ruta (experimento 3.3.1).

Los dos experimentos con los que se consiguieron estos resultados corresponden a los que utilizaron una penalización pareja para todos los clientes, excepto para la categoría de grandes clientes, la cual estuvo multiplicada por un factor de 10. Estos resultados fueron marginalmente superiores a los de los experimentos 3.2 y 3.3.1, en donde la penalización fue igual a todos los clientes. Sí hubo una mayor diferencia con los 3.1 y 3.1.1, que utilizaron una penalización basada en los tiempos de servicio. Esto se debe a que la categoría de minoristas es la con mayor cantidad de clientes, pero con menor tiempo de servicio, por lo que en este último caso no se prioriza respetar su asignación, frente a los otros clientes.

Por otro lado, se muestran los beneficios de utilizar una asignación de clientes a camiones basada en un criterio específico. Tanto en los experimentos 4.x como 5.x la base de conocimiento utilizado para incentivar la consistencia estuvo dada por la frecuencia de visitas de camiones a clientes durante el año, al resolver las instancias con el modelo de optimización; la consistencia del servicio aumento.

Para los experimentos 4.1.1 y 5.1.1, en los cuales solo se asignó un camión por cliente y una cantidad igual de clientes por camión; se obtuvo que un 62% de los clientes fueron servidos por el camión que tenían asignados, mientras que el 95% y 96% fue atendido por a lo más 5 camiones distintos respectivamente.

De igual forma que en los experimentos basados en las asignaciones de la Empresa, los que obtuvieron un mejor resultado con respecto a la consistencia del servicio fueron los modelos con aprendizaje, siendo levemente mejores los 4.x y 5.x al 3.x. Sin embargo, si no se considera el aprendizaje, o nuevas asignaciones de clientes, la asignación basada en la frecuencia de visitas del VRP sí obtiene resultados considerablemente mejores. El mejor de estos casos es el del experimento 5.1, donde un 59% de los clientes tienen un servicio consistente, mientras que en el 3.3 solo un 48% llegan a este nivel.

La diferencia de los resultados entre ambos modelos se hace más evidente al comparar los resultados de consistencia solo en la categoría de grandes clientes. Mientras que utilizando la asignación de clientes de la Empresa se obtuvo un 24% de consistencia (3.3.1) con aprendizaje de choferes, y 12% (3.3) sin aprendizaje, con la asignación mejorada se llegó hasta un 40% (4.1.1) y 32% (4.1) de consistencia con y sin aprendizaje respectivamente.

Con respecto a los dos últimos experimentos mencionados, se observa que, aunque son los mejores resultados para consistencia de un camión por cliente, estos utilizan una penalización igual para todos los clientes, sin priorizar la categoría deseada. Pero si se analiza cuántos clientes son servidos por más de un camión, se tiene que en el experimento 5.1.1, que sí prioriza la categoría de grandes clientes en la penalización, todos los clientes de la categoría son servidos por no más de 6 camiones distintas durante el año, mientras que en el 4.1.1 solo un 92% de los clientes son servidos por esta cantidad de camiones. De esta forma, se cumple

el objetivo planteado de aumentar la consistencia del servicio en la categoría de grandes clientes.

Finalmente, sobre el experimento 6, realizado con las asignaciones camiones-clientes de la Empresa, pero eliminando la restricción de ventanas de tiempo, se observa un aumento importante en la consistencia en general. Son servidos un 95% de los clientes por a los más 5 camiones diferentes, siendo este el mejor valor de consistencia alcanzado en un modelo con asignación de clientes a camiones sin optimizar, y sin aprendizaje. Esto nos lleva a inferir que la restricción de ventanas de tiempo es un factor importante en el diseño rutas consistentes, ya que parece afectarla considerablemente

Con respecto al costo en la función objetivo al incentivar la consistencia, en primer lugar, se observa que este aumenta considerablemente en los experimentos 3.x, esto es debido a que se incentiva la formación de rutas en base a la asignación de clientes de la empresa. Como se mostró en los resultados del objetivo (a), estas rutas tienen costos muy altos, y es evidente que, al incentivar replicarlas con el fin de aumentar la consistencia, el costo aumente.

Por otro lado, en los experimentos 4.x y 5.x, que utilizaron las asignaciones basadas en la frecuencia de visitas, se puede apreciar que el costo no aumenta ni disminuye de forma importante. Sin embargo, sí hay un leve aumento en las variantes 4.1 y 5.1; ambas tienen solo un vehículo asignado por cliente, por lo que es entendible que el costo aumente un poco al forzar la creación de rutas con clientes en donde no siempre es ideal su asignación en cuanto a costos. Por otro lado, se ve que en las variantes 4.3, 4.4, 5.3 y 5.4 se produce una disminución del costo; siendo estas variantes que tiene asignados 3 o 4 vehículos por cliente. Esta disminución de costos se puede explicar porque el modelo utilizado incentiva a asignar los clientes a las rutas que fueron asignados con mayor frecuencia en el CVRPTW, o sea las mejores rutas, y de esta forma, se utilizan menos iteraciones del algoritmo en formar las mejores rutas, dejando más iteraciones para mejorarlas aún más.

VIII. CONCLUSIONES

En el presente trabajo de tesis se abordó el problema operacional que enfrenta la logística de una importante empresa productora de alimentos, al distribuir producto a sus clientes, con sus restricciones horarias de ventanas de tiempo; para aprovechar de mejor forma la limitada flota disponible. Así, se implementó una variante del problema de ruteo de vehículos que se hiciera cargo de esta situación particular, siendo este el primer objetivo del trabajo.

Esta implementación se basó en la metaheurística simulated annealing, resolviendo un CVRPTW. Con respecto al primer objetivo, se concluye que la implementación de un modelo de ruteo puede traer grandes beneficios para la Empresa, ya que permite una mejor utilización de la flota disponible. Esto se traduce en una disminución aproximada de un 40% de la distancia recorrida en las rutas de despacho, y aún más importante, se puede aumentar el nivel de servicio en un 26% para los días de lata demanda, y entregar un 27% más de producto. En los días de baja demanda no se produce un aumento ya que la flota actual sí puede entregar todos los pedidos en dichos días, pero la disminución de distancia recorrida es más importante, llegando a un 44%.

Como se mencionó, la implementación anterior, consideró las restricciones de la operación, sin embargo, la Empresa también considera una preferencia a que los mismos clientes sean atendidos por los mismos camiones o choferes. De esta forma, en primer lugar, se mide el impacto que tiene una resolución del CVRPTW en la situación descrita, concluyendo que un modelo de ruteo que solo considere las restricciones de capacidad y de ventanas de tiempo implica bajos niveles de consistencia en el servicio. Siendo servidos solo el 12% de los clientes por un único camión. Así, en caso de la Empresa considere esta condición importante para la operación, es necesario utilizar otro tipo de modelos.

Por esto, se incorporaron incentivos a la consistencia en la función objetivo del CVRPTW, con el fin de aumentarla en la mayor medida posible, sin incorporarla como una restricción. Esto porque los modelos de ConVRP en el estado del arte solo son capaces de resolver instancias sin ventanas de tiempo, además de estar diseñados para planificar una serie de días seguidos, conociendo la demanda, y no para ser flexibles a más largo plazo sin conocerla.

De esta forma, se comprueba que al incorporar incentivos para la consistencia se puede llegar a altos niveles de esta, obteniendo hasta que un 58% de los clientes tengan un servicio consistente,

y que el 96% pueden ser atendidos por no más de 5 camiones diferentes. Los resultados anteriores fueron obtenidos con asignaciones de clientes a camiones sin ningún criterio de optimización, o sea, los que utilizaba la Empresa, asignados sin ningún criterio específico. Si se incorporan estos criterios para realizar nuevas asignaciones, el nivel de consistencia puede llegar a un 62%, tal como reflejan los resultados obtenidos. Por otro lado, al priorizar algún grupo de clientes específico, en el caso de este trabajo el de grandes cadenas, se tienen niveles de consistencia de un 24% y 40% respectivamente. Así se concluye que, para tener un ruteo consistente y costo eficiente, es necesario realizar una optimización previa. En esta optimización se deben asignar clientes a vehículos específicos, en base a ciertos criterios. En nuestro caso, se mostró que el criterio utilizado, basado en la frecuencia de visitas del VRP, es una alternativa para la asignación.

Finalmente, sobre el costo de la consistencia en la función objetivo, se tiene que este aumenta levemente al tratar de forzarla. Además, llama la atención que en algunos el costo promedio disminuyó. Si bien ambas observaciones pueden ser atribuibles a que el algoritmo no es determinístico, y en distintas corridas puede entregar mejores o peores resultados, también se puede considerar como otro indicador de que el algoritmo es mejorable. Esto porque en donde disminuyó, los experimentos utilizaron el conocimiento basado en la frecuencia de visitas del experimento 1, con más de un vehículo por cliente; de esta forma, se incentiva a reproducir parte de estas rutas, haciendo que el algoritmo tienda más rápido a una buena solución, y dando más tiempo para mejorarla.

Así, se concluye que se puede obtener una formulación del CVRPTW en la que se pueda aumentar la consistencia de una serie de rutas, con un enfoque distinto hasta el planteado en la literatura (ConVRP), ya que en dichos enfoques se planifica una serie de rutas para días seguidos, conociendo la demanda con anticipación, realizando una optimización off-line. En contraste, en nuestro enfoque se utiliza el conocimiento de rutas anteriores para tratar de repetir rutas y asignaciones en la mayor medida posible. Si bien así no se logran los altos niveles de consistencia que tiene la formulación clásica del ConVRP, sí se plantea como una buena alternativa para optimización on-line, en la que además se consideren ventanas de tiempo; acercándose a las necesidades de la Empresa.

IX. PROYECCIONES

La metodología empleada se concibe como un acercamiento a una formulación para el problema de ruteo de vehículos con consistencia, pero también con ventanas de tiempo, y en la que además no sea necesario conocer la demanda de una serie de días futuros. De esta forma se tiene un modelo en el que se pueda resolver el ruteo, incentivando la consistencia, pero con una mayor flexibilidad. Ya que se permite integrar en las rutas a nuevos clientes que no estén registrados con anterioridad, e incluso brindarles un servicio consistente a futuro, al considerar el aprendizaje o actualización permanente de las asignaciones de clientes a camiones o rutas.

La aplicación de esta formulación en un problema del mundo real consigue mantener los beneficios del ruteo de vehículos, junto con agregar una ventaja operacional al incentivar la consistencia del servicio, deseable para la Empresa. Lo que permitiría utilizarla como el núcleo de un software de apoyo para la toma de decisiones operacionales del área de logística.

Sin embargo, como se mencionó en capítulos anteriores, el algoritmo utilizado, es aún mejorable, lo que se refleja en que no siempre se llegue al mejor valor conocido en el caso de los problemas de benchmark, y que en algunos casos las soluciones de los experimentos con consistencia tengan un menor costo que las soluciones base del CVRPTW. Por lo que sería recomendable incluir mejoras en la implementación del simulated annealing para mejorar su efectividad.

Por otro lado, un aspecto importante de la formulación es la asignación de clientes rutas o camiones, ya que es esta la que da la consistencia que luego se trata de respetar a través de las penalizaciones. Dada la importancia mostrada para esta asignación, un trabajo futuro a realizar es investigar otras metodologías para obtener esta asignación. Ya que, como se explicó, en este trabajo se utilizó una asignación basada en la frecuencia de visitas a los clientes por distintas rutas, al resolver todas las instancias de un año con el CVRPTW. Luego, se resuelve el problema de asignación máxima para tener k rutas o camiones, con una cantidad igual de clientes cada uno, donde a cada ruta se asignan los clientes con mayor frecuencia de visitas posibles. Es en este último paso dónde se podrían incorporar mejoras a la metodología, basadas en la geografía del problema utilizado, ya que se presentan localidades más o menos cercanas al centro de distribución, y además con distintas densidades de clientes. Por lo que es posible que se encuentren mejoras al asignar distintas cantidades de clientes, basado en los criterios anteriores.

X. BIBLIOGRAFÍA

[1] G. Dantzig and J. Ramser (1959). The truck dispatching problem. Management Science, 6:80–91.

[2] J-F Cordeau, G. Desaulniers, J. Desrosiers, M.M. Solomon, F. Soumis (2000). The VRP with time Windows. Montreal: Groupe detudes et de recherche en analyse des decisions.

[3] Toth, P., & Vigo, D. (Eds.). (2014). Vehicle routing: problems, methods, and applications (Vol. 18). Siam.

[4] S. Ropke (2005). Heuristic and exact algorithms for vehicle routing problems (Tesis PHD). University of Copenhage, Denmark.

[5] J-F Cordeau, Gendreau, M., Laporte, G., J-Y Potvin, & Semet, F. (2002). A Guide to Vehicle Routing Heuristics. The Journal of the Operational Research Society, 53(5), 512-522.

[6] Clarke, G., & Wright, J. W. (1964). Scheduling of vehicles from a central depot to a number of delivery points. *Operations research*, *12*(4), 568-581.

[7] Fisher, M. L., & Jaikumar, R. (1981). A generalized assignment heuristic for vehicle routing. *Networks*, *11*(2), 109-124.

[8] Blum, C., & Roli, A. (2008). Hybrid metaheuristics: an introduction. In *Hybrid Metaheuristics* (pp. 1-30). Springer Berlin Heidelberg.

[9] F. Semet and E. Taillard (1993). Solving real-life vehicle routing problems efficiently using tabu search. Annals of Operations Research, 41:469-488.

[10] Gendreau, M., Hertz, A., & Laporte, G. (1994). A tabu search heuristic for the vehicle routing problem. *Management science*, *40*(10), 1276-1290.

[11] Cordeau, J. F., & Maischberger, M. (2012). A parallel iterated tabu search heuristic for vehicle routing problems. *Computers & Operations Research*, *39*(9), 2033-2050.

[12] Resende, M. G., & Ribeiro, C. C. (2010). Greedy randomized adaptive search procedures: Advances, hybridizations, and applications. In *Handbook of metaheuristics* (pp. 283-319). Springer US.

[13] Vincent, F. Y., Lin, S. W., Lee, W., & Ting, C. J. (2010). A simulated annealing heuristic for the capacitated location routing problem. *Computers & Industrial Engineering*, *58*(2), 288-299.

[14] Vidal, T., Crainic, T. G., Gendreau, M., & Prins, C. (2013). A hybrid genetic algorithm with adaptive diversity management for a large class of vehicle routing problems with time-windows. *Computers & operations research*, *40*(1), 475-489.

[15] Yu, B., & Yang, Z. Z. (2011). An ant colony optimization model: the period vehicle routing problem with time windows. *Transportation Research Part E: Logistics and Transportation Review*, *47*(2), 166-181.

[16] Krichen, S., Faiz, S., Tlili, T., & Tej, K. (2014). Tabu-based GIS for solving the vehicle routing problem. *Expert Systems with Applications*, *41*(14), 6483-6493.

[17] Vidal, T., Battarra, M., Subramanian, A., & Erdogan, G. (2015). Hybrid metaheuristics for the clustered vehicle routing problem. *Computers & Operations Research*, *58*, 87-99.

[18] Baños, R., Ortega, J., Gil, C., Márquez, A. L., & De Toro, F. (2013). A hybrid meta-heuristic for multi-objective vehicle routing problems with time windows. *Computers & Industrial Engineering*, *65*(2), 286-296.

[19] Bektas, T. (2006). The multiple traveling salesman problem: an overview of formulations and solution procedures. *Omega*, *34*(3), 209-219.

[20] Kovacs, A. A., Golden, B. L., Hartl, R. F., & Parragh, S. N. (2014). Vehicle routing problems in which consistency considerations are important: A survey. *Networks*, *64*(3), 192-213.

[21] Zhong, H., Hall, R. W., & Dessouky, M. (2007). Territory planning and vehicle dispatching with driver learning. *Transportation Science*, *41*(1), 74-89.

[22] Groër, C., Golden, B., & Wasil, E. (2009). The consistent vehicle routing problem. *Manufacturing & service operations management*, *11*(4), 630-643.

[23] Christofides, N. (1971). Fixed routes and areas for delivery operations. *International Journal of Physical Distribution*, *1*(2), 87-92.

[24] Gendreau, M., & Tarantilis, C. D. (2010). Solving large-scale vehicle routing problems with time windows: The state-of-the-art. Montreal: Cirrelt

XI. ANEXOS

Anexo 1: Script en python 2.7 de simulated annealing.

```
from __future__ import division
import math
import time
import matplotlib.pyplot as plt

from util import *
from heuristicas import *
from data_Empresa import *

######## NUMERO DE REPETICIONES DE SA ########
N = 1
###### PARAMETROS SIMULTED ANNEALING ########
#numero de iteraciones
nFinal = 100000
#temperatura inicial
Ts = 500
#definir tipo de cambio de temperatura, 1 = logaritmico, 2 = exponencial
temperatureChange = 1
#parametro para funcion de temperatura exponencial
epsilon = 0.00025
#porcentaje de sampleo en busqueda local
per = 0.1
################################################

################ FUNCIONES SA ##################
#funcion objetivo
def fo(p):
   return routing_cost(p, distances)

#funcion de creacion de inicial
def create():
   saving_list = get_savings(times)
   return clarke_and_wright(demand, saving_list, times, time_windows, truck_capacity,
np.float64(1))

#funcion de mutacion
def mutate(p):
   i = np.random.randint(4)
   return neighborhood(p, demand, distances, times, time_windows, truck_capacity, i, per)

#funcion dencambio de temperatura
```

```
def getTemperature(t):
    if temperatureChange == 1:
        if t < math.exp(1):
            T = Ts
        else:
            T = Ts/math.log(t)
    elif temperatureChange == 2:
        T = math.pow((1 - epsilon), t-1)*Ts
    return T


############# SIMULATED ANNEALING ##############
best_fo = bigM
for n in range(N):
    tStart = time.time()
    data = np.empty((nFinal+2,4))

    pCur = create()
    x = pCur

    t = 1
    while t <= nFinal:
        T = getTemperature(t)
        pNew = mutate(pCur)
        deltaE = fo(pNew) - fo(pCur)
        if deltaE < 0:
            pCur = pNew
            if fo(pCur) < fo(x):
                x = pCur
        else:
            if np.random.uniform(0,1) < math.exp(-deltaE/T):
                pCur = pNew
        t += 1

    tEnd = time.time()
    if fo(x) < best_fo:
        best_x = np.copy(x)
        best_fo = fo(x)
```

Anexo 2: Scrip de heuristicas, heuristicas.py.

```
#!/usr/bin/env python2
# -*- coding: utf-8 -*-
from util import *
from numba import jit
import numba as nb

bigM = 10000000

@jit('int64[:,:](float64[:], int64[:,:], float64[:,:], float64[:,:], int64, float64)', target='cpu',
nopython=True, cache=True)
def clarke_and_wright(demand, savings_list, times, time_windows, truck_capacity, lbd):
   #create empty array to save routes
   max_len_route = len(demand)+1
   routes = np.zeros((max_len_route-1,max_len_route), dtype=np.int64)
   s = 0
   node_in_routes = np.zeros(max_len_route-1, dtype=np.int64)
   while s < len(savings_list):
      if savings_list[s,0] == 0 or savings_list[s,1] == 0 or savings_list[s,0] == savings_list[s,1]:
         s += 1
         continue
      #unset vars
      combined = np.array([0], dtype=np.int64)
      found = False
      found_i = False
      found_j = False
      node_i_in = 0
      node_j_in = 0

      i = savings_list[s][0]
      j = savings_list[s][1]

      #First step : search nodes in current routes, if both nodes already are in a route unset the
opposite saving, if only one node is on the route, save it.
      for r in xrange(len(routes)):
         if routes[r,0] == 0: continue
         route = routes[r]
         if found_i == False:
            if index2(decode_route(route),i) >= 0:
               found_i = True
            if found_i: node_i_in = r
         if found_j == False:
            if index2(decode_route(route),j) >= 0:
               found_j = True
            if found_j: node_j_in = r

      if found_i or found_j:
         found = True #at least one node was found
```

```
        if found_i and found_j: #both nodes are allready conected in a route
            s += 1
            continue
    #Second step: both nodes founds then combine routes if nodes not interior
    if found_i and found_j:
        if interior_node(j, routes[node_j_in]) or interior_node(i, routes[node_i_in]):
            x=2
            #routes without 0 on the beginning and end
            route_i = decode_route(routes[node_i_in])
            route_j = decode_route(routes[node_j_in])
            ipos = np.where(route_i == i)[0][0]
            jpos = np.where(route_j == j)[0][0]
            ipos=0
            jpos=0
            if ipos == 0 and jpos == len(route_j)-1: #i at the beginning and j at the end
                combined = np.concatenate((route_j, route_i))
            elif jpos == 0 and ipos == len(route_i)-1:
                combined = np.concatenate((route_i, route_j))
            if len(combined) > 2 and check_restrictions(combined, demand, times, time_windows,
truck_capacity): #set route
                routes = remove_route(routes[node_i_in], routes)
                routes = remove_route(routes[node_j_in], routes)
                routes = insert_route(combined, routes)
                s += 1
                continue

        else: #dosen't fit restricctions
            s += 1
            continue
    #Third step: one node was found, add if possible
    #case found node j
    elif found_j and found_i == False:
        if routes[node_j_in][0] == j: #i will be added only if j is the first visit of truck
            route_j = decode_route(routes[node_j_in])
            combined = np.concatenate((np.array([i], dtype=np.int64), route_j))
            if check_restrictions(combined, demand, times, time_windows, truck_capacity): #set route
                routes = remove_route(routes[node_j_in], routes)
                routes = insert_route(combined, routes)
                s += 1
                continue
        else: #dosen't fit restricctions
            s += 1
            continue

    #case found node i
    elif found_i and found_j == False:
        if routes[node_i_in][len(decode_route(routes[node_i_in]))-1] == i: #j will be added only if i is
the last visit of truck
            route_i = decode_route(routes[node_i_in])
```

```
            combined = np.concatenate((route_i, np.array([j], dtype=np.int64)))
            if check_restrictions(combined, demand, times, time_windows, truck_capacity): #set route
               routes = remove_route(routes[node_i_in], routes)
               routes = insert_route(combined, routes)
               s += 1
               continue
         else: #dosen't fit restricctions
            s += 1
            continue

      #Fourth step: if not found, create new route and continue
      if found == False:
         node_in_routes[i]=1
         node_in_routes[j]=1
         route = np.array([i,j], dtype=np.int64)
         if check_restrictions(route, demand, times, time_windows, truck_capacity):
            routes = insert_route(route, routes)
            s += 1
            continue
         else: #dosen't fit restricctions
            s += 1
            continue
      s += 1

   #Fifth step: create single routes with all remaining client nodes
   for node in xrange(len(demand)):
      found = False
      for r in routes[:,:max_len_route-1].ravel():
         if node == r:
            found = True
            break
      if not found:
         route = np.array([node], dtype=np.int64)
         if check_restrictions(route, demand, times, time_windows, truck_capacity):
            routes = insert_route(route, routes)
         else:
            continue
   return routes

@jit(target='cpu', nopython=True, cache=True)
def relocate(routes, node_pos, new_pos, demand, times, time_windows, truck_capacity, infeasible =
False):
   node = routes[node_pos]
   route1 = decode_route(routes[node_pos[0]])
   route1 = np.concatenate((route1[:node_pos[1]] ,route1[(node_pos[1]+1):]))
   #case 1: both nodes ar on the same route
   if node_pos[0] == new_pos[0] and len(route1) > 1:
      route1 = np.concatenate((route1[:new_pos[1]], np.array([node], dtype=np.int64),
route1[new_pos[1]:]))
```

```
      if check_restrictions(route1, demand, times, time_windows, truck_capacity) or infeasible:
         new_routes = np.zeros((1,len(routes[0,:])), dtype=np.int64)
         new_routes[0] = encode_route(route1, len(routes[0,:]))
         return new_routes
   #case 2: different routes
   elif node_pos[0] != new_pos[0]:
      route2 = decode_route(routes[new_pos[0]])
      route2 = np.concatenate((route2[:new_pos[1]], np.array([node], dtype=np.int64),
route2[new_pos[1]:]))
      if ((check_restrictions(route1, demand, times, time_windows, truck_capacity) and
check_restrictions(route2, demand, times, time_windows, truck_capacity)) or infeasible) and
len(route2) < len(routes[0,:]):
         new_routes = np.zeros((2,len(routes[0,:])), dtype=np.int64)
         new_routes[0] = encode_route(route1, len(routes[0,:]))
         new_routes[1] = encode_route(route2, len(routes[0,:]))
         return new_routes

#Swap the position of two nodes
@jit(target='cpu', nopython=True, cache=True)
def exchange(routes, pos1, pos2, demand, times, time_windows, truck_capacity, infeasible = False):
   if pos1[0] == pos2[0]:
      new_routes = np.zeros((1,len(routes[0,:])), dtype=np.int64)
   else:
      new_routes = np.zeros((2,len(routes[0,:])), dtype=np.int64)
   new_routes[0] = routes[pos1[0]]
   new_routes[0][pos1[1]] = routes[pos2]
   if pos1[0] == pos2[0]:
      new_routes[0][pos2[1]] = routes[pos1]
   else:
      new_routes[1] = routes[pos2[0]]
      new_routes[1][pos2[1]] = routes[pos1]
   if check_routes_restrictions(new_routes, demand, times, time_windows, truck_capacity) or
infeasible:
      return new_routes
   else:
      return None

@jit(target='cpu', nopython=True, cache=True)
def two_opt(routes, pos1, pos2, demand, times, time_windows, truck_capacity, infeasible = False):
   route1 = decode_route(routes[pos1[0]])
   if pos1[0] == pos2[0] and len(route1) > 3:
      a = min(pos1[1],pos2[1])
      b = max(pos1[1],pos2[1])
      new_route = np.concatenate((route1[:a], route1[a:][:(b-a)][::-1], route1[b:]))
      if check_restrictions(new_route, demand, times, time_windows, truck_capacity) or infeasible:
         new_routes = np.zeros((1,len(routes[0,:])), dtype=np.int64)
         new_routes[0] = encode_route(new_route, len(routes[0,:]))
         return new_routes
   if pos1[0] != pos2[0]:
```

```
        route2 = decode_route(routes[pos2[0]])
        new_route1 = np.concatenate((route1[:pos1[1]], route2[:pos2[1]][::-1]))
        new_route2 = np.concatenate((route2[pos2[1]:][::-1], route1[pos1[1]:]))
        if ((check_restrictions(new_route1, demand, times, time_windows, truck_capacity) and
check_restrictions(new_route2, demand, times, time_windows, truck_capacity)) or infeasible) and
(len(new_route1) < len(routes[0,:]) and len(new_route2) < len(routes[0,:])):
            new_routes = np.zeros((2,len(routes[0,:])), dtype=np.int64)
            new_routes[0] = encode_route(new_route1, len(routes[0,:]))
            new_routes[1] = encode_route(new_route2, len(routes[0,:]))
            return new_routes

@jit(target='cpu', nopython=True, cache=True)
def two_opt_as(routes, pos1, pos2, demand, times, time_windows, truck_capacity, infeasible = False):
    route1 = decode_route(routes[pos1[0]])
    if pos1[0] == pos2[0]:
        return None
    route2 = decode_route(routes[pos2[0]])
    new_route1 = np.concatenate((route1[:pos1[1]], route2[pos2[1]:]))
    new_route2 = np.concatenate((route2[:pos2[1]], route1[pos1[1]:]))
    if ((check_restrictions(new_route1, demand, times, time_windows, truck_capacity) and
check_restrictions(new_route2, demand, times, time_windows, truck_capacity)) or infeasible) and
(len(new_route1) < len(routes[0,:]) and len(new_route2) < len(routes[0,:])):
        new_routes = np.zeros((2,len(routes[0,:])), dtype=np.int64)
        new_routes[0] = encode_route(new_route1, len(routes[0,:]))
        new_routes[1] = encode_route(new_route2, len(routes[0,:]))
        return new_routes

@jit(target='cpu', nopython=True, cache=True)
def cross_exchange(routes, pos1, pos2, demand, times, time_windows, truck_capacity, infeasible =
False):
    if pos1[0] == pos2[0] or pos1[1] == pos1[2] or pos2[1] == pos2[2]:
        return None
    route1 = decode_route(routes[pos1[0]])
    route2 = decode_route(routes[pos2[0]])
    new_route1 = np.concatenate((route1[:(min(pos1[1],pos1[2]))],
route2[(min(pos2[1],pos2[2])):(max(pos2[1],pos2[2]))], route1[(max(pos1[1],pos1[2])):]))
    new_route2 = np.concatenate((route2[:(min(pos2[1],pos2[2]))],
route1[(min(pos1[1],pos1[2])):(max(pos1[1],pos1[2]))], route2[(max(pos2[1],pos2[2])):]))
    if ((check_restrictions(new_route1, demand, times, time_windows, truck_capacity) and
check_restrictions(new_route2, demand, times, time_windows, truck_capacity)) or infeasible) and
(len(new_route1) < len(routes[0,:]) and len(new_route2) < len(routes[0,:])):
        new_routes = np.zeros((2,len(routes[0,:])), dtype=np.int64)
        new_routes[0] = encode_route(new_route1, len(routes[0,:]))
        new_routes[1] = encode_route(new_route2, len(routes[0,:]))
        return new_routes

@jit(target='cpu', nopython=True, cache=True)
def neighborhood(routes, demand, distances, times, time_windows, truck_capacity, visits,
clients_penalty, LS = 0, per = 1, intra=False, first=False, infeasible = False):
```

```
    best_improve = -bigM
    found = False
    rute1_indices = sample_routes(routes, per)
    for route1_pos in rute1_indices:
        if intra:
            route2_indices = np.array([route1_pos], dtype=np.int64)
        else:
            route2_indices = sample_routes(routes, per)
        for route2_pos in route2_indices:
            for node1_pos in xrange(len_route(routes[route1_pos])):
                for node2_pos in xrange(len_route(routes[route2_pos])):
                    improve = -bigM*2
                    pos1 = (route1_pos, node1_pos)
                    pos2 = (route2_pos, node2_pos)
                    if  pos1 == pos2:
                        continue
                    if LS == 0:
                        move = relocate(routes, pos1, pos2, demand, times, time_windows, truck_capacity,
infeasible)
                    if LS == 1:
                        move = exchange(routes, pos1, pos2, demand, times, time_windows, truck_capacity,
infeasible)
                    if LS == 2:
                        move = two_opt_as(routes, pos1, pos2, demand, times, time_windows, truck_capacity,
infeasible)
                    if LS == 3:
                        move = two_opt(routes, pos1, pos2, demand, times, time_windows, truck_capacity,
infeasible)
                    if move is not None:
                        # print "move Tipo: {}, dtype {}, shape {}".format(type(move), move.dtype, move.shape)
                        improve = move_improve(routes, (route1_pos,route2_pos), move, distances, visits,
clients_penalty)

                    if (first and improve>0) or (improve > best_improve):
                        tls = LS
                        best_pos = (route1_pos, route2_pos)
                        best_improve = improve
                        best_move = move
                        found = True

                    if (first and improve>0):
                        N = np.copy(routes)
                        return apply_move(N, best_pos, best_move)

    if found:
        N = np.copy(routes)
        return apply_move(N, best_pos, best_move)
    else:
        return routes
```

Anexo 3: Script de utilidades, util.py.

```
#!/usr/bin/env python2
# -*- coding: utf-8 -*-
from numba import jit, vectorize
import numpy as np
import pandas as pd

@jit('int64(int64[:])', target='cpu', nopython=True, cache=True)
def len_route(route):
   return route[len(route)-1]

#take route in "matrix format " ([0,1,2,..,0,-1,-1,..]) and return [0,1,2,..,0]
@jit('int64[:](int64[:])',target='cpu', nopython=True, cache=True)
def decode_route(route):
   return route[:len_route(route)]

#reverse decode_route
@jit('int64[:](int64[:],int64)', target='cpu', nopython=True, cache=True)
def encode_route(route, max_len_route):
   if len(route) < max_len_route:
      res = np.zeros(max_len_route, dtype=np.int64)
      res[0:len(route)] = route
      for i in xrange(0,max_len_route):
         if res[i] == 0: break
      res[max_len_route-1] = i
   elif len(route) == max_len_route:
      res = route
   return res


@jit('int64[:,:](int64[:,:])',target='cpu', nopython=True, cache=True)
def valid_routes(routes):
   return routes[np.where(routes[:,0] != 0)]

@jit('int64(int64[:,:])', target='cpu', nopython=True, cache=True)
def count_routes(routes):
   return len(valid_routes(routes))

@jit('int64[:](int64[:,:])',target='cpu', nopython=True, cache=True)
def valid_routes_index(routes):
   return np.where(routes[:,0] != 0)[0]

#insert route on routes matrix, encoded or decoded route
@jit('int64[:,:](int64[:],int64[:,:])', target='cpu', nopython=True, cache=True)
def insert_route(route, routes):
   for i in xrange(len(routes)):
      if routes[i][0] == 0: break
#    i = np.where(routes[:,1] == 0)[0][0]
```

```
    routes[i] = encode_route(route, len(routes[0,:]))
    return routes

#get depot node on numpy array
@jit('int64[:]()', target='cpu', nopython=True, cache=True)
def depot():
    return np.array([0], dtype=np.int64)

@jit('float64(int64[:],float64[:,:],float64[:,:])', target='cpu', nopython=True, cache=True)
def route_time(route, times, time_windows):
    min_time = time_windows[:,1]
    max_time = time_windows[:,2]
    service_time = time_windows[:,3]
    total_time = times[0,route[0]]
    if total_time < min_time[route[0]]:
        total_time = min_time[route[0]] + service_time[route[0]]
    else:
        total_time += service_time[route[0]]
    i = -1
    for i in xrange(len(route)-1):
        total_time += times[route[i],route[i+1]]
        if total_time < min_time[route[i+1]]:
            total_time = min_time[route[i+1]] + service_time[route[i+1]]
        else:
            total_time += service_time[route[i+1]]
        if route[i+1] == 0:
            break
    #last node in case of not coded route
    if route[i+1] != 0:
        total_time += times[route[i+1],0]
    return total_time

@jit('float64(int64[:],float64[:,:])', target='cpu', nopython=True, cache=True)
def route_distance(route, distances):
    total_distance = distances[0,route[0]]
    i = -1
    for i in xrange(len(route)-1):
        total_distance += distances[route[i],route[i+1]]
        if route[i+1] == 0:
            break
    #last node in case of not coded route
    if route[i+1] != 0:
        total_distance += distances[route[i+1],0]
    return total_distance

@jit('float64(int64[:],float64[:])',target='cpu', nopython=True, cache=True)
def route_demand(route,demand):
    total_demand = 0
    for i in xrange(len(route)):
```

```
        total_demand += demand[route[i]]
        if route[i+1] == 0:
            break
    return total_demand

#@jit('float64(int64[:,:],float64[:,:],int64[:],int64[:,:],boolean)', target='cpu', nopython=True,
cache=True)
@jit(target='cpu', nopython=True, cache=True)
def routing_cost(routes, distances, routes_indices, visits, clients_penalty, penalty=True):
    cost = 0
    indices = valid_routes_index(routes)
    for i in indices:
        cost += route_distance(routes[i], distances)
    if penalty:
        cost += con_penalty(routes,routes_indices,indices,visits,clients_penalty)
    return cost

@jit('boolean(int64[:], float64[:], float64[:,:], float64[:,:], int64)', target='cpu', nopython=True,
cache=True)
def check_restrictions(route, demand, times, time_windows, truck_capacity):
    min_time = time_windows[:,1]
    max_time = time_windows[:,2]
    service_time = time_windows[:,3]
    if len(route) == 0: return True
    current_time = times[0,route[0]]
    route_demand = demand[route[0]]
    #capacity constraint depto to fisrt node
    if current_time > max_time[route[0]] or route_demand > truck_capacity:
        return False
    #TW constraint depot to first node
    if current_time < min_time[route[0]]:
        current_time = min_time[route[0]] + service_time[route[0]]
    else:
        current_time += service_time[route[0]]
    for i in xrange(1,len(route)):
        #capacity constraint
        route_demand += demand[route[i]]
        if route_demand > truck_capacity:
            return False
        #TW constraint
        current_time += times[route[i-1],route[i]]
        if current_time > max_time[route[i]]:
            return False
        if route[i] == 0:
            break
        if current_time < min_time[route[i]]:
            current_time = min_time[route[i]] + service_time[route[i]]
        else:
            current_time += service_time[route[i]]
```

```
    #last node in case of not coded route
    if route[i] != 0 and current_time + times[route[i],0] > max_time[0]:
        return False
    return True

@jit('boolean(int64[:,:], float64[:], float64[:,:], float64[:,:], int64)', target='cpu', nopython=True,
cache=True)
def check_routes_restrictions(routes, demand, times, time_windows, truck_capacity):
    for i in xrange(len(routes)):
        if check_restrictions(routes[i], demand, times, time_windows, truck_capacity) == False:
            return False
    return True

@jit('float64[:,:](float64[:,:], float64)',target='cpu', nopython=True, cache=True)
def get_savings_base(times, lbd = 1):
    n_costumers = len(times)
    savings = np.zeros((n_costumers,n_costumers), dtype=np.float64)
    for i in xrange(n_costumers):
        for j in xrange(n_costumers):
            if j != 0 and i != j:
                savings[i,j] = times[0,i] + times[0,j] - lbd*times[i,j]
            else:
                savings[i,j] = -1000000000
    return savings

def get_savings(times, lbd = 1):
    n_costumers = len(times)
    savings = get_savings_base(times, lbd).ravel()
    savings_index_raveled = np.argsort(savings)[::-1]
    savings_index = np.dstack(np.unravel_index(savings_index_raveled, (n_costumers,
n_costumers)))[0]

    return savings_index

@jit('int64(int64[:], int64)',target='cpu', nopython=True, cache=True)
def index2(ar, a):
    for i in xrange(ar.shape[0]):
        if ar[i] == a:
            return i
    return -1

#returno FALSE is a node is at the start os the END of a route, TRUE is is interior
@jit('boolean(int64,int64[:])',target='cpu', nopython=True, cache=True)
def interior_node(node, route):
    route2 = decode_route(route)
    interior_nodes = route2[2:-2]
    if index2(interior_nodes,node)>=0:
        return True
    else:
```

```
        return False

@jit('int64(int64[:,:], int64, int64)', target='cpu', nopython=True, cache=True)
def index(ar, a, b):
    for i in xrange(ar.shape[0]):
        if ar[i][0] == a and ar[i][1] == b:
            return i
    return -1

#get the position of a node on the routes matrix
@jit(target='cpu', nopython=True, cache=True)
def node_pos(node, routes):
    pos = np.where(routes == node)
    return (pos[0][0], pos[1][0])

def print_routes(routes):
    for route in valid_routes(routes):
        print decode_route(route)

@jit(target='cpu', nopython=True, cache=True)
def sample_nodes(nodes, per):
    return np.random.choice(nodes, size=np.int64(round(len(nodes)*per)), replace=False)

@jit(target='cpu', nopython=True, cache=True)
def sample_routes(routes, per):
    routes_index = valid_routes_index(routes)
    return np.random.choice(routes_index, size=np.int64(round(len(routes_index)*per)), replace=False)

@jit(target='cpu', nopython=True, cache=True)
def no_depot_route(route):
    route = decode_route(route)
    return route[np.where(route!=0)]

@jit(target='cpu', nopython=True, cache=True)
def add_tabu(move, route_pos, moved_nodes, tabu_list):
    for i in xrange(0, len(moved_nodes)):
        j = np.where(tabu_list[:,1] == -1)[0][0]
        tabu_list[j][:] = np.array([move, route_pos, moved_nodes[i]], dtype=np.int64)
    return tabu_list

#@jit(target='cpu', nopython=True, cache=True)
def check_tabu(move, route_pos, nodes, tabu_list):
    for i in xrange(0, len(nodes)):
        if any(np.equal(tabu_list,[move, route_pos, nodes[i]]).all(1)) == True:
            return False
    return True

@jit(target='cpu', nopython=True, cache=True)
def move_improve(routes, pos, move, distances,visits,penalty):
```

```
    a = route_distance(routes[pos[0]], distances)
    if pos[0] != pos[1]:
        a += route_distance(routes[pos[1]], distances)
    a += con_penalty(routes, pos, valid_routes_index(routes), visits, penalty)
    b = routing_cost(move, distances, np.array(pos,dtype=np.int64), visits, penalty, False)
    indices = valid_routes_index(routes)
    for i in xrange(len(move)):
        driver = index2(indices, pos[i])
        b += route_penalty(move[i], visits[driver], penalty)
    return a-b

@jit(target='cpu', nopython=True, cache=True)
def apply_move(routes, pos, move):
    for i in xrange(len(move)):
        if len_route(move[i]) == 0: routes = remove_route_pos(pos[i], routes)
        else: routes[pos[i]] = encode_route(move[i], len(routes[0,:]))
    return routes

#remove route from routes, given route index on matrix
@jit(target='cpu', nopython=True, cache=True)
def remove_route_pos(index, routes):
    routes[index] = np.zeros(len(routes[0,:]), dtype=np.int64)
    return routes

#remove route from routes matrix, devien the encoded or decoded route
@jit(target='cpu', nopython=True, cache=True)
def remove_route(route, routes):
    route = encode_route(route, len(routes[0,:]))
    #find row i where is route in routes, and delete routes[i]
    for index in xrange(len(routes)):
        equal = True
        if routes[index][0] != route[0]:
            equal = False
        if equal:
            break;
    routes = remove_route_pos(index, routes)
    return routes

#for a set of routes, rename node number to customer ID
def id_routes(routes, nodes_id):
    routes1 = np.copy(routes)
    for i in valid_routes_index(routes):
        for j in xrange(len(decode_route(routes[i]))):
            routes1[i,j] = nodes_id[routes1[i,j]]
    return routes1

@jit(target='cpu', nopython=True, cache=True)
def remove_node(node_pos,routes):
    route = decode_route(routes[node_pos[0]])
```

```
    route = np.concatenate((route[:node_pos[1]],route[node_pos[1]+1:]))
    routes[node_pos[0]] = encode_route(route, len(routes[node_pos[0]]))
    return routes

def instance_customers_data(customers_id):
    nodes = np.append(np.zeros(1, dtype=np.int64), customers_id)
    distance_matrix = pd.read_csv("data/Empresa distance
matrix.csv",delimiter=";",header=None).as_matrix()
    distances = distance_matrix[nodes,:][:,nodes]
    time_matrix = pd.read_csv("data/Empresa time matrix.csv",delimiter=";",header=None).as_matrix()
    times = time_matrix[nodes,:][:,nodes]
    #ventas de tiempo
    tw_data = pd.read_csv("data/Empresa TW.csv",delimiter=";",header=None).as_matrix()
    time_windows = np.empty((len(nodes),4))
    customer_type = pd.read_csv("data/Empresa
customers.csv",delimiter=";",header=None).as_matrix()
    for i in xrange(len(nodes)):
        time_windows[i] = tw_data[int(customer_type[nodes][:,1][i])]
    #convertir tw a minutos
    time_windows[:,1:3] = time_windows[:,1:3]*60
    return distances, times, time_windows

def customers_data():
    distance_matrix = pd.read_csv("data/Empresa distance
matrix.csv",delimiter=";",header=None).as_matrix()
    time_matrix = pd.read_csv("data/Empresa time matrix.csv",delimiter=";",header=None).as_matrix()
    #ventas de tiempo
    tw_data = pd.read_csv("data/Empresa TW.csv",delimiter=";",header=None).as_matrix()
    customer_type = pd.read_csv("data/Empresa
customers.csv",delimiter=";",header=None).as_matrix()
    time_windows = np.empty((len(customer_type),4))
    for i in xrange(len(customer_type)):
        time_windows[i] = tw_data[int(customer_type[:,1][i])]
    #convertir tw a minutos
    time_windows[:,1:3] = time_windows[:,1:3]*60
    return distance_matrix, time_matrix, time_windows

@jit(target='cpu', nopython=True, cache=True)
def con_penalty(routes, routes_indices, indices, visits, penalty):
    cost = 0
    for r in xrange(len(routes_indices)):
        i = routes_indices[r]
        cost += route_penalty(routes[i], visits[index2(indices,i)], penalty)
    return cost

@jit('float64(int64[:], int64[:], float64[:])', target='cpu', nopython=True, cache=True)
def route_penalty(route, visits, penalty):
    #version original con cantidad de visitas totales
    cost = np.float64(0)
```

```
    for r in xrange(len(decode_route(route))):
        if visits[route[r]] == 0:
            cost += penalty[r]
    return cost

#clientes de categoria dada. 1=MINORISTAS, 2=SUPERMERCADO, 3=MAYORISTA, 5=HORECA
# 6=DISTRIBUIDORES, 7=GRANDES CADENAS, 50=OTROS.
def clients_by_category(categories):
    client_type = pd.read_csv("data/Empresa customers.csv",delimiter=";",header=None).as_matrix()
    clients = np.array([],dtype=np.int64)
    for i in categories:
        clients = np.append(clients, np.where(client_type[:,1] == i))
    return clients
```