Inteligencia Artificial

Estado del Arte: Truck and Trailer Routing Problem

Matías Vargas

8 de diciembre de 2022

Evaluación

Resumen (5 %):	_____
Introducción (5 %):	_____
Definición del Problema (10 %):	_____
Estado del Arte (35 %):	_____
Modelo Matemático (20 %):	_____
Conclusiones (20 %):	_____
Bibliografía (5 %):	_____
Nota Final (100 %):	_____

Resumen

La finalidad del presente documento es realizar un análisis del problema TTRP, el cual nace como evolución del problema VRP, considerando aplicaciones más cercanas a la realidad. Este consiste en abastecer un conjunto de clientes con una flota de camiones, los cuales pueden tener un trailer acoplado y el objetivo principal es que cada cliente sea abastecido una única vez y el costo incurrido por la flota sea mínimo. Este problema tiene diversas aplicaciones en la vida real, generalmente es utilizado en problemas donde se deben abastecer clientes y los caminos para llegar a ellos son de difícil acceso, por tanto se decide desacoplar el trailer en una ubicación conveniente y continuar solo con el camión. Al ser un problema interesante en la industria, varios investigadores han decidido abordado y en este documentos nos centramos en los diferentes enfoques utilizados entre el año 2002 y 2015.

1. Introducción

1.1. Propósito y Estructura del documento

Un problema recurrente en la industria de transporte de cargas, es abastecer la demanda de diferentes clientes en forma óptima. La finalidad es gastar lo menos posible en insumos de transportes (bencina, mantenciones, costos de transportes en general) y además seleccionar el tipo adecuado de transporte, en base a la demanda que se desea abastecer. En base a este contexto surge el problema *Truck and Trailer Routing Problem* (TTRP), el cual será definido en profundidad con la siguiente estructura: En la sección 2, se define el problema y la estructura básica con sus diferentes variantes. En la sección 3, se realizara un estado del arte acerca de

TTRP, comenzando desde el año 2002 por Chao[3] hasta el año 2015 realizado por Isis[8], presentando las diferentes heurísticas de construcción y reparación de diversos investigadores. En la sección 4, se define en profundidad el modelo matemático del problema. Finalmente, la sección 5 se encuentran los detalles de la representación que se utilizó para resolver el problema y en la sección 6 la descripción del algoritmo, con su explicación a través de pseudocódigos.

1.2. Descripción del problema y Motivación

El problema *Truck and trailer routing problem* (TTRP) es introducido por Chao[3] en año 2002. Este nace como una variante del problema *Vehicle Routing problem* (VRP), siendo un acercamiento más tangible de la realidad. El problema TTRP consiste en en una flota de vehículos compuesto por m_k camiones y m_l trailers (compartimientos) que deben servir a un conjunto de clientes y además la cantidad de camiones debe ser mayor a la de trailers. A causa de diversos problemas con las rutas (caminos de difícil acceso) es que se distinguen diferentes tipos de clientes y rutas.
La motivación para el estudio de TTRP es el lugar central que ocupa en la administración y logística de distribución, teniendo diversos usos industriales. Gerdessen[4] presenta dos aplicaciones de TTRP. La primera es la distribución de productos lácteos por parte de la industria holandesa, en la que muchos clientes se encuentran en ciudades de alto tráfico y estacionamiento limitado. Maniobrar vehículos en estos ambientes es complicado, por lo que deciden dejar el trailer afuera de la ciudad y continuar la entrega de los clientes sin el trailer.
La segunda aplicación se basa en la entrega de alimentos para animales en diversos graneros, en donde los caminos y puentes son muy estrechos, por lo que hay diferentes tipos de vehículos para transitar.

2. Definición del Problema

El problema que busca resolver TTRP es satisfacer la demanda de todos los clientes con una determinada flota de camiones, los cuales deben empezar y terminar en el mismo depósito. Cada camión debe tener su propia ruta asociada (diferentes tipos de rutas) y los clientes que serán visitados (diferentes tipos de clientes), con la restricción que todos deben ser visitados solo una vez. El objetivo del problema es minimizar la distancia total recorrida o el costo incurrido de la flota. Para entender el problema hay que comprender las diferentes componentes que se pueden visualizar en la figura [1] extraída del paper de Ulrich [2], estas son las siguientes:

2.1. Variables del problema

- En primer lugar se debe solucionar un problema de asignación y la variable binaria utilizada consiste en asignarle a cada cliente i un camión de la flota j.
- El segundo lugar se debe solucionar el problema de TSP para cada camión y la variable binaria representa si el camión va del cliente i al cliente j.

2.2. Tipos de clientes

- Cliente de vehículo: Puede ser servido por el camión con o sin el trailer.
- Cliente de camión: Solo puede ser servido con el camión sin el trailer.

2.3. Tipo de rutas

- Ruta de camión pura: El camión satisface la demanda sin usar el trailer. Se pueden servir clientes de tipo camión o vehículo.

- Ruta de vehículo pura: El camión sirve los clientes con el trailer acoplado. Solo clientes de vehículo son servidos.
- Ruta de vehículo completa: El camión debe desacoplar al menos una vez el trailer y estacionarlo en un cliente de tipo vehículo (este trayecto es la ruta principal). Posteriormente, el camión continúa sirviendo un subconjunto de clientes en una sub ruta. Después de servir a todos, el camión debe volver al cliente de vehículo donde estaciono el trailer y finalmente puede realizar otra sub ruta o bien anclar el compartimiento y continuar con la ruta principal.

2.4. Particionamiento de la ruta de vehículo completa

- Ruta principal: Es la secuencia de clientes servidos por el vehículo completo.
- Sub ruta: Es la secuencia que comienza cuando el camión aparca el trailer en un cliente de vehículo, el cual se conoce como la raíz de la sub ruta.

2.5. Restricciones

- El cliente puede ser servido solo una vez.
- El camión inicia en un depósito y debe finalizar en él.
- Las sub rutas pueden sobrepasar la capacidad del camión, solo en el caso que este permitido la transferencia de productos.
- La longitud de la ruta recorrida por el vehículo puede estar limitada.
- La demanda de ruta vehículo pura y completa no debe superar la capacidad del camión más el compartimiento.
- La sub ruta debe comenzar y finalizar en el mismo cliente de vehículo, antes de continuar la ruta principal.
- La cantidad de camiones debe ser mayor a la de trailers.

2.6. Paramétros del problema

- n es la cantidad de clientes.
- m_k es la cantidad de camiones.
- m_l es la cantidad de trailers.
- $m_k - m_l$ es la cantidad de rutas de camión puro.
- c_{ij} es el costo asociado a la distancia euclideana entre el cliente i y el cliente j.
- q_i es la demanda del cliente i.
- Q_k y Q_l es la capacidad del camión y trailer respectivamente.
- El problema puede ser representado mediante un grafo $G = \{V, E\}$, en donde $V = \{1, 2, 3, ..., n\}$ es el conjunto de clientes (0 el depósito) y E es el conjunto de arcos entre los nodos clientes.

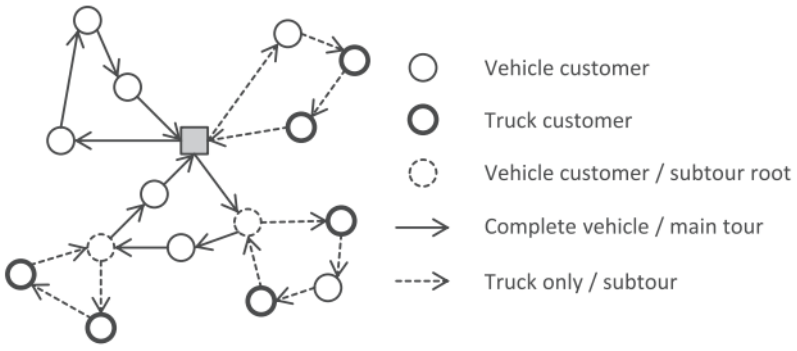
2.7. Variantes del problema

- Considerar ventanas de tiempos: Esta variante (TTRPTW) considera que los clientes solo aceptan ser servidos durante un periodo de tiempo específico.
- Caso en que no se puede desacoplar el trailer y además puede haber más de un deposito (MDTTRP).
- Los vehículos no son necesarios que vuelvan al depósito (OTTRP)
- Considerar transferencia de productos entre el compartimiento y el camión
  - Modelo de Chao [3]: Considera transferencia de productos siempre y cuando la recarga no requiera de herramientas o dispositivos que no esté a bordo.
  - Modelo de Gerdessen [4]: Se consideran que los productos se pueden tranfererir, ya que son livianos o líquidos.
  - Modelo de Ulrich [2]: No considera transferencia de productos.

2.8. Problemas Relacionados

- Vehicle routing problem (VRP): Consiste en una flota de vehículos homogéneo que se distribuyen en diferentes rutas para servir diferentes clientes. Todos deben iniciar en un único depósito. El objetivo es a cada vehículo proveer una secuencia de visitas en la cual todos los clientes sean servidos y la distancia total de viaje por la flota sea mínima.
- Partial Accessibility Constraint VRP (PACVRP): Se asume que todos los camiones disponibles son utilizados y el número de compartimientos necesita ser determinado. Pueden haber cambios en los costos, al considerarse el sueldo de todos los conductores.
- VRP con compartimiento (VRPT): Planteado por Gerdessen [4], en donde se realizan dos importantes suposiciones. La primera es que todos los clientes poseen una demanda unitaria. En segundo lugar, cada compartimiento puede ser estacionado solo una vez, es decir, una ruta de vehículo completa con exactamente un sub-tour. Gracias a esto último, no es necesario considerar costos de estacionamiento y desacoplamiento del compartimiento.

Figura 1: Esquema del problema TTRP



3. Estado del Arte

En las siguientes subsecciones, se realizara un análisis completo sobre el estado del arte del problema TTRP, con las diferentes heurísticas y métodos que se han utilizado para la resolución del problema, estas serán presentadas en orden cronológico.

3.1. Descent improvement steps

En el año 2002 Chao [3] desarrollo las bases del problema, considerando una heurística de mejora de soluciones que es conocida como *Descent Improvement steps* para el problema *TTRP*. Consiste en que los clientes son asignados a una de los tres tipos de rutas, a través de resolver un problema de asignación. Posteriormente, se intenta cambiar a los clientes a una nueva ruta, con el objetivo de convertir una solución que previamente era infactible en una factible. La estrategia es considerar una penalidad θ que mida el grado de infactibilidad de este cambio. Se considera una penalidad de la ruta R (θ_R), la cual es computada como $\theta_R = max(0, \alpha_R - \Gamma_R)$, en donde α_R es la demanda de la ruta R y Γ_R es la capacidad del camión de R.

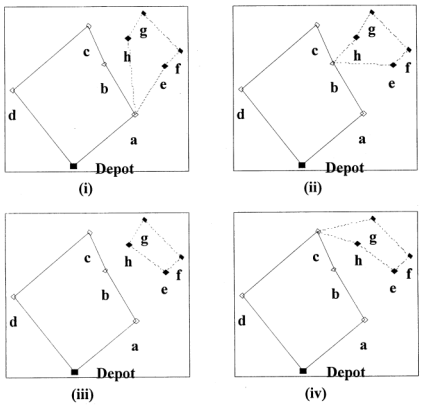
1. *One-point descent movement*: La finalidad es mover un cliente de una ruta a otra. Si el movimiento candidato decrementa la penalidad con o sin un aumento de distancia, o bien disminuye la distancia sin aumentar la penalidad, entonces se ejecuta inmediatamente el cambio. Hay dos movimientos que deben ser excluidos: El primero es mover un cliente de camión a un tour principal perteneciente a una ruta de vehículo completa, o bien a una ruta de vehículo pura. La segunda es mover un cliente de vehículo, que es el estacionamiento de un sub-tour (cliente raíz).

2. *Two-point descent exchange*: Dos clientes de dos rutas diferentes pueden ser intercambiados. Considerándose las mismas restricciones que el caso anterior.

3. *Sub-tour root-rexning step*: En los casos anteriores los nodos raíces no cambiaron su posición, sin embargo es posible que el cambio genere una mejor solución. En este paso se intenta intercambiar los nodos raíces y hacer una posible re secuencia de las sub rutas, eliminando el primer y último arco que llega al nodo raíz y probar otras combinaciones, con otro nodo raíz. Este proceso se puede apreciar en la imagen [2].

Chao resolvio 21 instancias del problema [3]. Estas se consolidaron como las bases para los investigadores posteriores, siendo los casos de comparación.

Figura 2: Paso de refinamiento de un nodo raíz



3.2. Tabu Search

Esta es una heurística de mejora basada en realizar búsquedas locales, siendo ampliamente utilizado en problemas de optimización combinatorial. Sirve para guiar la búsqueda y evitar quedarse atrapado en óptimos locales, moviéndose en cada iteración de una solución S a la mejor solución vecina posible S'. Para evitar las búsquedas cíclicas, las soluciones visitadas recientemente que cumplen un determinado atributo se memorizan en una lista tabú (restricción tabú) durante un número de iteraciones. Las soluciones vecinas de S que contienen tal atributo se consideran temporalmente tabú o prohibidas, a menos que cumplan con el llamado criterio de aspiración.

Dentro del mismo año 2002, Chao [3], en su paper utiliza el método *Tabu Search*, proponiendo dos restricciones tabús: La primera es *FTB*, la cual prohíbe que un cliente vuelva a una ruta del que fue eliminado en las últimas π iteraciones. La segunda restricción es OTB (objective-based value) y hace uso del concepto *Deterministic annealing*. Este último consiste en solo considerar soluciones con valores de la función objetivo que están dentro de una cantidad específica (desviación) del mejor valor de la función objetivo hasta el momento. Una solución candidata con un valor de la función objetivo mayor que el mejor valor de la función objetivo más una desviación estará prohibida como restricción tabú. Por otra parte, usa un criterio de aspiración que anula la restricción tabú cada vez que una solución candidata produce un valor de función objetivo que es menor que el mejor valor actual.

3.3. T-Cluster and T-Sweep

En el año 2006, Scheuerer [7] propone la utilización de dos heurísticas de construcción, para la mejora de los resultados obtenidos por Chao.
En primer lugar se encuentra T-Cluster, este es considerado como un procedimiento de inserción secuencial basado en grupos, donde las rutas se construyen cliente a cliente hasta la utilización total de la capacidad del vehículo. Se inicializa una nueva ruta con el cliente sin ruta más alejado del depósito y el vehículo no utilizado que tiene la capacidad total máxima. En el caso de que sea un vehículo completo, si el cliente inicial es de vehículo, entonces se inserta en el recorrido principal y si es de camión se insertara en un nuevo sub recorrido. El siguiente cliente para ser insertado en la ruta es minimizando la siguiente ecuación: $e(k) = c_{ku} + c_{kf} - \pi c_{0k}$, donde k especifica el cliente en consideración, f es el cliente más cercano enrutado en esta ruta y 0 el depósito. El término c_{ku} mide la distancia del cliente inicial de la ruta u y el cliente en

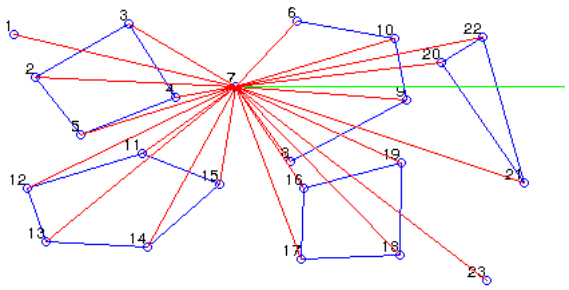
Figura 3: Instancias de problema TTRP diseñadas por Chao

Problem number	Original[a]	Customers		Trucks		Trailers		Ratio of demand to capacity
		v.c.	t.c.	Number	Capacity	Number	Capacity	
1		38	12					
2	CMT1	25	25	5	100	3	100	0.971
3		13	37					
4		57	18					
5	CMT2	38	37	9	100	5	100	0.974
6		19	56					
7		75	25					
8	CMT3	50	50	8	150	4	100	0.911
9		25	75					
10		113	37					
11	CMT4	75	75	12	150	6	100	0.931
12		38	112					
13		150	49					
14	CMT5	100	99	17	150	9	100	0.923
15		50	149					
16		90	30					
17	CMT11	60	60	7	150	4	100	0.948
18		30	90					
19		75	25					
20	CMT12	50	50	10	150	5	100	0.903
21		25	75					

consideración k. Esta distancia se debe mantener mínima para garantizar la compactitud de la ruta. El segundo término c_{kf} asegura que está cerca del "borde"de la ruta actual. Finalmente el térmico πc_{0k} es conocido como el término de diversificación, en donde la variación del parámetro π resultara en la selección de una nueva estrategia.

En segundo lugar se encuentra T-Sweep presentado en la figura [4], es un heurística basada en el clásico algoritmo *Sweep* atribuido a Gillett y Miller [1]. Esta técnica de barrido es utilizada para constuir soluciones factibles girando un rayo centrado en el depósito e incluyendo gradualmente a los clientes en una ruta de vehículos. Se inicializa una nueva ruta con el vehículo no utilizado que tiene la capacidad total máxima, siempre que se alcance la capacidad total del vehículo o la restricción de longitud de la ruta. Si no hay más vehículos disponibles, se permite que la última ruta se vuelva inviable.

Figura 4: Ejemplo de T-Sweep



Finalemente, Scheuerer [7] también utiliza la heurística de *Tabu Search*, pero con nuevas mejoras. En la figura [5] se puede ver que las soluciones encontradas por Scheuerer son mejores para las 21 instancias de Chao, en donde $c(s^*)$ representa la distancia total, $q(s^*)$ la sobrecapacidad total y T el tiempo total, en segundos.

Figura 5: Comparación de soluciones entre Chao y Scheurer

ID	T-Cluster[a]			T-Sweep[b]			Chao constr.[c,e]		Chao descent[c,e]		$c(s^{**})$
	$c(s^*)$	$q(s^*)$	T[d]	$c(s^*)$	$q(s^*)$	T[d]	$c(s^*)$	$q(s^*)$	$c(s^*)$	$q(s^*)$	
1	651.87	0.0	0.33	644.80	0.0	0.30	657.15	9.6	646.02	0.0	564.68
2	697.51	0.0	0.27	722.46	0.0	0.23	739.04	13.9	739.90	0.0	612.75
3	766.25	0.0	0.25	797.65	0.0	0.23	785.54	16.8	774.78	0.0	618.04
4	979.79	0.0	0.45	901.78	26.0	0.56	937.82	26.0	943.47	0.0	798.53
5	1037.50	0.0	0.42	1035.76	32.0	0.55	1108.87	22.9	1130.85	0.0	839.62
6	1173.11	0.0	0.41	1171.99	52.0	0.55	1174.17	32.1	1236.69	0.0	933.26
7	904.77	0.0	1.09	901.14	0.0	1.88	937.31	14.1	906.31	0.0	830.48
8	965.90	0.0	1.00	1005.99	0.0	1.63	1004.45	18.5	971.60	0.0	878.36
9	1081.21	0.0	0.94	1099.88	0.0	1.45	1156.50	45.6	1106.66	0.0	934.47
10	1167.38	0.0	1.83	1150.42	0.0	4.84	1232.10	33.6	1159.78	0.0	1039.07
11	1274.67	0.0	1.94	1288.49	0.0	3.94	1422.41	38.0	1288.74	0.0	1094.11
12	1438.11	0.0	1.69	1443.00	0.0	3.77	1578.79	34.0	1453.82	0.0	1155.13
13	1485.67	0.0	2.72	1482.02	0.0	7.91	1624.16	35.3	1481.40	0.0	1287.18
14	1611.99	0.0	2.67	1658.55	0.0	7.42	1760.51	37.1	1624.96	0.0	1353.08
15	1748.31	0.0	2.66	1892.89	0.0	7.42	2105.02	33.6	1858.87	0.0	1457.61
16	1055.23	0.0	1.98	1383.57	0.0	3.53	1288.48	10.3	1267.87	0.0	1002.49
17	1117.22	0.0	1.64	1416.14	0.0	3.31	1314.09	9.4	1261.17	0.0	1042.35
18	1216.24	0.0	1.77	1614.11	0.0	2.91	1383.19	10.8	1366.21	0.0	1129.16
19	874.04	0.0	0.86	919.59	0.0	1.34	1146.74	22.0	969.96	0.0	813.50
20	950.72	0.0	0.78	972.76	0.0	1.24	1144.96	24.0	1140.47	0.0	848.93
21	1009.38	0.0	0.75	1096.08	0.0	1.31	1263.70	57.0	1174.43	0.0	909.06

3.4. Simulated Annealing

Shih-Wei Lin [6] en el año 2009 propone la utilización de una heurística llamada *Simulated annealing* para resolver el problema de TTRP. Esta es una heuristica de búsqueda local capaz de escapar de un óptimo local, aceptando en pequeña probabilidad peores soluciones durante ciertas iteraciones. El concepto de annealing hace alusión a lograr un cercano mínimo global imitando el enfriamiento lento producido en el proceso físico de recocido de los metales. Este comienza con una solución inicial aleatoria y en cada iteración el algoritmo toma una nueva solución de la vecindad predefinida en la solución actual. El valor de la función objetivo de la nueva solución es comparado con la solución actual y en caso que sea mejor, se reemplaza la solución actual.
Shih-Wei Lin [6] mezcla este método con una búsqueda aleatoria de vecindad que presenta varios tipos de movimientos. Para la inserción establece seleccionar un cliente aleatorio i de la solución S e insertarlo en la posición precedente de otro cliente aleatorio j de la solución S. Por otro lado, el intercambio es seleccionando aleatoriamente el cliente x y j de la solución S, para luego intercambiarlos. Finalmente cambiar el tipo de vehículo es seleccionando aleatoriamente un cliente de vehículo y cambiar el servicio a tipo camión o vice versa.

Es importante destacar que Lin en el año 2011 resuelve instancias para el problema TTRPTW (TTRP con ventanas de tiempo), siendo el primero en resolverlo.

En la figura[6] se puede apreciar la comaparación entre Lin, Chao y Scheuerer. En donde pudo obtener 17 mejores soluciones de los 21 problemas y 4 instancias continuaron con mejor solución para Scheurer, sin embargo el tiempo de ejecución de Lin fue el mejor en todos los casos. $c(s^*)$ representa la distancia total y T el tiempo de ejecución en segundos.

Figura 6: Comparación entre Lin, Chao y Scheurer

ID	Chao		Scheuerer ($\lambda = 15\,000$)			SA heuristic			$c(s^{**})^k$
	Min $c(s^*)^a$	T^b	Min $c(s^*)^c$	Avg $c(s^*)^d$	T^e	Min $c(s^*)^f$	Avg $c(s^*)^g$	T^h	
1	565.02	4.19	566.80	567.98	9.51	566.82	568.86	6.80	564.68[i,j]
2	662.84	5.22	615.66	619.35	9.60	612.75	617.48	6.67	611.53[j]
3	664.73	6.50	620.78	629.59	11.24	618.04	620.50	5.59	618.04[i,j]
4	857.84	7.53	801.60	809.13	18.49	808.84	817.71	16.32	798.53[i,j]
5	949.98	7.06	839.62	858.98	15.16	839.62	858.95	14.42	839.62[i,j]
6	1084.82	7.96	936.01	949.89	18.62	934.11	942.60	13.65	930.64 [j]
7	837.80	16.43	830.48	832.91	33.60	830.48	838.50	24.96	830.48[i,j]
8	906.16	11.11	878.87	881.26	25.66	875.76	882.70	24.03	872.56[j]
9	1000.27	10.18	942.31	955.95	30.47	912.64	921.97	21.75	912.02[j]
10	1076.88	21.72	1039.23	1052.65	60.94	1053.90	1074.38	63.61	1039.07[j]
11	1170.17	17.10	1098.84	1107.47	56.17	1093.57	1108.88	60.33	1093.57[j]
12	1217.01	20.27	1175.23	1184.58	63.71	1155.44	1166.59	51.70	1154.73[j]
13	1364.50	42.34	1288.46	1296.33	165.41	1320.21	1340.98	119.56	1287.18[i]
14	1464.20	25.96	1371.42	1384.13	132.06	1351.54	1367.91	113.75	1347.40[j]
15	1544.21	24.62	1459.55	1488.71	154.10	1436.78	1454.91	93.87	1425.87[j]
16	1064.89	14.56	1002.49	1003.00	43.14	1004.47	1007.26	41.46	1002.49[i]
17	1104.67	13.74	1042.35	1042.79	33.73	1026.88	1035.23	38.81	1026.20[j]
18	1202.00	12.52	1129.16	1141.94	31.78	1099.09	1110.13	31.34	1098.15[j]
19	887.22	16.13	813.50	813.98	28.84	814.07	823.01	29.58	813.30[j]
20	963.06	10.09	848.93	852.89	24.57	855.14	859.06	28.47	848.93[j]
21	952.29	9.57	909.06	914.04	26.84	909.06	915.38	24.03	909.06[i,j]
Avg.	1025.74	14.51	962.40	970.84	47.32	958.06	968.24	39.56	953.53
ARPD	7.57%[l]		0.93%[l]	1.82%[m]		0.48%[l]	1.54%[m]		0.00

3.5. GRASP and VNS

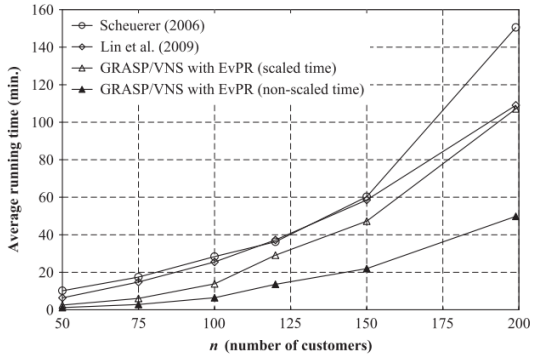
En el año 2011, Villegas [5], introduce una forma para resolver el problema, basado en *Greedy Randomized Adaptive Search Procedure* (GRSAP), *Variable Neighborhood Search* (VNS). En primer lugar GRSAP, es un metaheurística de inicio múltiple sin memoria, que consiste en iteraciones compuestas para la construcción de una solución aleatoria codiciosa y mejoras iterativas de la misma a través de una búsqueda local. Por otra parte, VNS es un método de búsqueda que explora vecindarios distantes de la solución actual a través de dos fases, la primera es de descenso para encontrar un óptimo local y la segunda es una fase de perturbación para salir del valle.

En al figura [7] se pueden apreciar las comparaciones entre Villegas y los investigadores anteriores. El estudio de Villegas encontró 11 mejores soluciones.
En la figura [8] se pueden apreciar la mejora en los tiempos de computación, para instancias de 50 clientes hasta 200.

Figura 7: Comparación entre Villegas, Lin, Chao y Scheurer

Chao [7] (tabu search)		Scheuerer [45] (tabu search)				Lin et al. [32] (simulated annealing)				Caramia and Guerriero [5] (math. prog. heuristic)		GRASP/VNS + EvPR			
Avg. Cost	Gap	Best cost	Gap	Avg. Cost	Gap	Best cost	Gap	Avg. Cost	Gap	Cost	Gap	Best cost	Gap	Avg. Cost	Gap
565.02	0.06	566.80	0.38	567.98	0.59	566.82	0.38	568.86	0.74	566.80	0.38	**564.68**	0.00	565.99	0.23
662.84	8.39	615.66	0.68	619.35	1.28	612.75	0.20	617.48	0.97	620.15	1.41	**611.53**	0.00	614.23	0.44
664.73	7.56	620.78	0.44	629.59	1.87	**618.04**	0.00	620.50	0.40	632.48	2.34	**618.04**	0.00	618.04	0.00
857.84	7.43	801.60	0.38	809.13	1.33	808.84	1.29	817.71	2.40	803.32	0.60	**798.53**	0.00	803.51	0.62
949.98	13.14	**839.62**	0.00	858.98	2.31	**839.62**	0.00	858.95	2.30	842.50	0.34	**839.62**	0.00	841.63	0.24
1084.82	16.57	936.01	0.58	949.89	2.07	934.11	0.37	942.60	1.29	938.18	0.81	940.59	1.07	961.47	3.31
837.80	0.88	**830.48**	0.00	832.91	0.29	**830.48**	0.00	838.50	0.97	832.56	0.25	**830.48**	0.00	830.48	0.00
906.16	3.85	878.87	0.72	881.26	1.00	875.76	0.37	882.70	1.16	878.87	0.72	**872.56**	0.00	876.21	0.42
1000.27	9.68	942.31	3.32	955.95	4.82	912.64	0.07	921.97	1.09	980.42	7.50	914.23	0.24	918.45	0.71
1076.88	3.64	1039.23	0.02	1052.65	1.31	1053.90	1.43	1074.38	3.40	1060.41	2.05	1046.71	0.74	1050.11	1.06
1170.17	7.02	1098.84	0.50	1107.47	1.29	1093.57	0.02	1108.88	1.42	1170.70	7.07	**1093.37**	0.00	1100.95	0.69
1217.01	5.61	1175.23	1.99	1184.58	2.80	1155.44	0.27	1166.59	1.24	1178.34	2.26	**1152.32**	0.00	1158.88	0.57
1364.50	6.01	1288.46	0.10	1296.33	0.71	1320.21	2.57	1340.98	4.18	1288.46	0.10	1298.89	0.91	1305.83	1.45
1464.20	9.32	1371.42	2.39	1384.13	3.34	1351.54	0.91	1367.91	2.13	1372.52	2.48	**1339.36**	0.00	1354.04	1.10
1544.21	8.69	1459.55	2.73	1488.71	4.79	1436.78	1.13	1454.91	2.41	1470.21	3.48	1423.91	0.22	1437.52	1.18
1064.89	6.22	**1002.49**	0.00	1003.00	0.05	1004.47	0.20	1007.26	0.48	1004.69	0.22	**1002.49**	0.00	1003.07	0.06
1104.67	7.65	1042.35	1.57	1042.79	1.62	1026.88	0.07	1035.23	0.88	1042.35	1.57	1042.46	1.58	1042.61	1.60
1202.00	9.46	1129.16	2.82	1141.94	3.99	1099.09	0.09	1110.13	1.09	1129.16	2.82	1113.07	1.36	1118.63	1.86
887.22	9.09	813.50	0.02	813.98	0.08	814.07	0.09	823.01	1.19	813.50	0.02	813.50	0.02	819.81	0.80
963.06	13.44	**848.93**	0.00	852.89	0.47	855.14	0.73	859.06	1.19	**848.93**	0.00	860.12	1.32	860.12	1.32
952.29	4.76	**909.06**	0.00	914.04	0.55	**909.06**	0.00	915.38	0.70	**909.06**	0.00	**909.06**	0.00	909.06	0.00

Figura 8: Gráfico de tiempos comparativo entre Villegas, Lin, Scheuerer



3.6. LS-ABHC, LS-RRT and LNS-RRT

En el año 2012, Ulrich [2] propone un enfoque de dos fases. En primer lugar, crear una solución inicial factible basada en búsqueda de vecindarios, *Local search* y *Large Neigborhood Search* (LNS). La segunda fase, se consolida en que los movimientos son aplicados en base a dos metaurísticas de control, *Record-to-Record Travel* (RRT) y *Attribute Based Hill Climber* (ABHC).

1. *Local Search*: Se basa en pequeños modificaciones a la solución, siendo su fuerte la intensificación de la búsqueda. Se consideran una serie de movimientos, como, intercambiar o relocalizar. Se puede habilitar la transferencia de sub-tours completos considerando la selección de un nuevo cliente raíz del sub-tour y también se puede cambiar la selección de clientes dentro del sub-tour. Otro tipo de cambio es clientes de vehículo completo a clientes de camión.

2. *Large neighborhood search*: El concepto de neighborhood se basa en arruinar y recrear. En el problema de TTRP, se aplica el paso de arruinar cuando se remueven una cantidad de clientes de una ruta, los cuales no son servidos temporalmente y el paso de recreación es cuando se re insertan esos clientes de nuevo en la solución con una heurística de inserción barata. En LNS el paso de destrucción es en base a la selección de clientes que son similares respecto a sus distancias.

3. *Record-to-record travel*: Se basa en utilizar el costo de la mejor solución encontrada hasta ahora, con la finalidad de definir la "aceptabilidad"de la solución. El vecino seleccionado (aleatoriamente) es aceptado si no es peor que la mejor solución encontrada hasta el momento por una desviación estándar.

4. Attribute based hill climber: Es un tipo de *Tabu Search*, que utiliza atributos genéricos para especificar vecinos no tabú, el cual debe ser especializado para el dominio de cada problema.

La resolución de Ulrich [2] logro encontrar 18 mejores nuevas soluciones en comparación a Villegas [5], encontrándose entre los mejores resultados del momento.

3.7. Fuzzy Constraints

En el año 2015, Torres [8] resuelve el problema con un enfoque de *Fuzzy Constraints*. La finalidad es tener un marcador de tolerancia de violación de las restricciones, este marcador permite que las restricciones sean satisfechas tanto como sean posibles. También menciona un enfoque paramétrico, definiendo un valor α que representa el marcador de nivel de satisfacción, en donde todas las restricciones son acotadas a este valor. Probando con diferentes α se encuentra con un conjunto de soluciones.

3.8. Tendencia

Los métodos propuestos por [2] y Villegas [5] son los que estan funcionando mejor actualmente y son el estado del arte. Por un lado Ulrich, tiene 18 mejores soluciones y utiliza las heurísticas *LS-ABHC, LS-RRT and LNS-RRT*. Por otro lado, Villegas tiene 4 mejores soluciones utilizando las heurísticas de *GRASP and VNS*.

4. Modelo Matemático

El modelo matemático que se presentara es planteado por el investigador Chao [3]. Se debe solucionar un problema de asignación relajado, en donde a cada cliente se le asigna un camión de la flota que cumplirá su demanda.

Paramétros:

- n es la cantidad de clientes.
- m_k es la cantidad de camiones.
- m_l es la cantidad de trailers.
- q_i es la demanda del cliente i.
- Q_l es la capacidad del trailer l.
- Q_k es la capacidad del camión k.
- $i = 1, 2, ..., n$. Representa el cliente número i.
- $j = 1, 2, ..., m_k$. Representa el camión número j.
- $d_{ij} = c_{0i} + c_{is_j} - c_{0s_j}$ representa el costo de asignarle al cliente i el camión j. Por otra parte, s_j se conoce como un cliente semilla para el camión j. La forma de obtenerlo es a través de realizar un proceso iterativo. En la primera iteración, la semilla es el cliente más lejano del depósito, en la segunda iteración el cliente semilla es el más lejano a las semillas precedentes y el depósitos. Se repite este proceso 10 veces encontrando diferentes soluciones iniciales, la última es la mejor.

Variables:

- $x_{ij} = \{0, 1\}$. Es la variable binaria que representa si al cliente i se le asigna el camión j.

Función objetivo:

$$min \sum_{i=1}^{n} \sum_{j=1}^{m_k} d_{ij} x_{ij} \tag{1}$$

Restricciones:

$$s.t. \sum_{j=1}^{m_k} x_{ij} = 1, \quad i = 1, 2, ..., n \tag{2}$$

$$\sum_{i=1}^{n} q_i x_{ij} \leq Q_l + Q_k, \quad j = 1, 2, ..., m_l \tag{3}$$

$$\sum_{i=1}^{n} q_i x_{ij} \leq Q_k, \quad j = m_l + 1, ..., m_k \tag{4}$$

$$0 \leq x_{ij}, \quad i = 1, 2, ..., n, \quad j = m_l + 1, ..., m_k \tag{5}$$

- La ecuación (1) plantea minimizar el costo total de asignamiento.

- Ecuación (2) establece la restricción que ningún cliente tenga asignado más de un camión.
- La restricción (3) fuerza que la máxima demanda de una ruta vehículo pura o completa sea menor o igual que la suma de la capacidad del camión y el trailer.
- La restricción (4) fuerza que la maxima demanda de cada ruta de camión debe ser menor o igual a la capacidad del camión.
- La restricción (5) fuerza a que los x_{ij} tomen valores en el intervalo cerrado $[0, 1]$
- El espacio de búsqueda es 2^{n*m_k}.

Chao[3] menciona que pueden haber clientes que tienen asignados valores fraccionarios, en ese caso hay que aproximar el valor a 1. Esto puede generar soluciones infactibles en base a la capacidad de los camiones, sin embargo debe ser solucionado con la heurística de mejora. Para este paso la única finalidad es asignar a los camiones los diferentes clientes.

Luego de resolver el problema de asignación, se debe construir las rutas de los camiones con los clientes asignados. Esta parte del problema se soluciona como el problema del vendedor viajero (TSP).

Paramétros:

- c_{ij} es el costo de ir del cliente i al cliente j.
- Q es el conjunto de recorridos posibles.

Variables:

- $x_{ij} = \{0, 1\}$. Es la variable binaria que representa si el camión va del cliente i al cliente j.

Función objetivo:

$$min \sum_{i=1}^{n} \sum_{j \neq i, j=1}^{n} c_{ij} x_{ij} \tag{6}$$

Restricciones:

$$\sum_{i=1, i \neq j}^{n} x_{ij} = 1, \quad j = 1, ..., n \tag{7}$$

$$\sum_{j=1, j \neq i}^{n} x_{ij} = 1, \quad i = 1, ..., n \tag{8}$$

$$\sum_{i \in Q} \sum_{j \neq i, j \in Q} x_{ij} \leq |Q| - 1, \quad \forall Q \subset \{1, ..., n\}, \quad |Q| \geq 2 \tag{9}$$

- La función objetivo en (6) busca minimizar el costo total de la ruta.
- La restricción (7) y (8) establecen que un cliente solo puede ser visitado una vez (del nodo cliente entra un arco y sale un único arco)
- La restricción (9) establece que no deben haber sub-recorridos, debe ser la unión de un único recorrido.
- El espacio de búsqueda es $(n-1)!/2$.

Las rutas de camión y vehículo pura son tratados rápidamente como TSP. En las rutas de vehículo completas, se construye primero la ruta principal y luego el sub recorrido, en donde se puede anidar a otro sub recorrido o comenzar uno nuevo uniéndolo a un cliente de la ruta principal.

5. Representación

Para la representación se utilizaron estructuras de C++ y vectores que son equivalentes a arreglos dinámicos con la ventaja de tener varios métodos disponibles para ser utilizados. A continuación se puede ver las estructuras para la asignación de rutas, crear la ruta de tsp y una estructura espacial para las subrutas.

```
typedef struct ASSIGNMENT_ROUTE{
    float id_semilla;
    float tipo_transporte;
    float demanda_total;
    float cantidad_demandada;
    float capacidad_total;
    float capacidad_disponible;
    vector<int> clientes;

} Route;

typedef struct ROUTETSP{
  vector<int> clientes;
  string type;
  float distance;
} RouteTSP;

typedef struct TruckSub{
  vector<int> id_clients;
  int id_ruta;
  float distance;
} TruckSub;
```

Al momento de resolver el problema de asignación se creara un vector de estructuras *assigment_route* que contiene los clientes que fueron asignados a cada camión, estos clientes deben satisfacer la capacidad del camión y tráiler según corresponda (cabe mencionar que solo estan los clientes asignados pero aún no se inicia una ruta óptima). La variable tipo_transporte es 0 si esa ruta no tiene un tráiler acoplado y 1 en caso contrario. También hay variables para guardar la demanda total que se ha satisfecho y la capacidad disponible que tiene el camión para abastecer al momento que se busca asignarle nuevos clientes.

Luego de ser asignados los clientes a una ruta se debe establecer el camino óptimo resolviendo un problema de *TSP*. Para esto, habrá un vector de estructuras *routetsp* que contiene la secuencia de los clientes, sin embargo, solo se consideran maintours en rutas completas o rutas vehículo puro y además variable distancia, guarda la distancia total de la ruta.

Finalmente, se crea un vector de estructuras *TruckSub* que se encarga de guardar los clientes de las subrutas y tiene como variable un id_ruta para reconocer a que ruta pertenece.

Como se puede apreciar, estas representaciones son bastante intuitivas para establecer la relación con el modelo matemático, en donde se pueden manejar restricciones fácilmente, como que un cliente se puede encontrar una única vez en la lista de clientes o que podemos guardar la relación capacidad disponible del camión para saber si podrá satisfacer un nuevo cliente que se busca asignarle, de esta forma satisfacemos las restricciones de oferta/demanda.

6. Descripción del algoritmo

Para resolver el problema TTRP se utilizó el enfoque propuesto por Chao [3]. En primer lugar se debe resolver el problema de asignación y para esto, debemos crear semillas (cada semilla es un cliente particular de una ruta) y asignarle una a cada camión. El proceso de generación de semillas es el siguiente:

- La primera semilla es la más lejana del depósito.
- La segunda semilla es la más lejana del deposito y de la primera semilla.
- La semilla n es la más lejana del depósito y de las n-1 semillas precedentes.

Para encontrar las semillas, en primer lugar debemos crear una matriz de distancias de dimensiones $(n+1)$x$(n+1)$ (n es la cantidad de clientes y +1 porque se incluye las distancias del depósito), esta matriz contiene las distancias entre todos los clientes y además es simétrica. Después de ser asignada cada semilla a cada ruta, se deben comenzar a asignar los clientes. Para esto se creó un algoritmo greedy que busca asignar a cada cliente la ruta de semilla s_j que tenga menor costo d_{ij}. En nuestro problema el costo es simplemente la distancia y este valor se calcula como:

$$d_{ij} = c_{0i} + c_{is_j} - c_{0s_j}$$

- d_{ij} costo de asignarle al cliente i el camión j.
- s_j es el cliente semilla para el camión j.
- c_{0i} es el costo del depósito al cliente i.
- c_{is_j} es el costo del cliente i a la semilla s_j.
- c_{0s_j} es el costo del depósito a la semilla s_j.

También se debe calcular una matriz de distancias de c_{is_j} (de los clientes a las semillas), para que el algoritmo greedy pueda seleccionar la menor distancia en esta matriz, la cual tendrá dimensiones de n x nCamiones.
A continuación se detallará el algoritmo en pseudocódigo:

Algorithm 1 Algoritmo greedy para asignación de rutas

```
// Cada nodo representa un cliente con propiedades de (id,x,y,demanda,tipo)
Variable listaNodos: CrearNodos(inputfile)
Variable matrizDistancias: CalcularDistanceMatrix()
Variable listaSemillas: [∅]
// Crear una semilla a cada camión
for numero en rango(cantidadCamiones) do
  CalcularSemilla(listaSemillas,matrizDistancias)
end for
// Matriz de distancia entre los clientes y las semillas
Variable matrizDij: CalcularDijMatrix(matrizDistancias, listaSemillas)
For each semilla in listaSemillas do
   InicializarRuta()
for cliente in listaNodos do
  encontrar semilla(ruta) que produce menor dij para el cliente
  if camión de esa ruta tiene capacidad disponible then
    agregar cliente a la ruta de ese camión
  else if camión de esa ruta no tiene capacidad disponible y quedan trailers disponibles then
    agregar trailer al camión
    agregar cliente a la ruta de ese camión
  else
    buscar una nueva semilla(ruta)
  end if
end for =0
```

Una vez resuelto el problema de asignación, es necesario crear la secuencia de las rutas y resolver el problema *TSP*. Para realizar esto, se creó un algoritmo greedy que en cada iteración busca el cliente más cercano al actual para continuar la secuencia. Como se mencionó en la definición del problema hay tres tipos de rutas (camión puro, vehículo puro y vehículo completa) y nuestro algoritmo debe controlar esos casos cuando va creando las secuencias. Esto es detallado a continuación y en el segundo pseudocódigo:

- Es tipo de ruta 0, si al camión no se le agrego tráiler cuando se resolvió el problema de asignación, eso quiere decir que los clientes pueden ser satisfechos con el camión sin el tráiler, siendo una ruta de **camión puro**. En este caso simplemente el algoritmo greedy crea la secuencia TSP, buscando al cliente más cercano del nodo actual.
- Es tipo de ruta 1, si al camión se le agrego el tráiler cuando se resolvió el problema de asignación. En esta situación hay dos casos posibles:
  - Todos los clientes asignados son de vehículo y no hay ninguno de camión. En este caso nos encontramos ante el tipo de ruta **vehículo pura**. El algoritmo greedy crea la secuencia TSP, buscando al cliente más cercano del nodo actual.
  - Hay clientes de camión y de vehículo. En este caso nos encontramos con el tipo de ruta **vehículo completa**. El algoritmo greedy se debe encargar de lo siguiente:
    - Crear un main tour con todos los clientes de vehículo de la misma forma que en los casos anteriores.
    - Crear subtours de la siguiente forma: se itera sobre los clientes de camión y si es el primero se debe buscar cual es el cliente más cercano presente en el main tour. Para los siguientes, se pregunta cual cliente es más cercano si uno del main tour, o alguno de los subtours ya creados. Para el primer caso simplemente se crea una subruta y para el segundo se agrega a la secuencia después del cliente más cercano.

Algorithm 2 Algoritmo greedy para crear secuencia de la ruta

```
Variable listaRutasTSP: [∅]
Variable listaSubtours: [∅]
Variable listaRutasAsignadas: greedyAssigment()
for ruta in listaRutasAsignadas do
  Variable listaClientesCamión: BuscarClientesCamión()
  Variable listaClientesVehiculo: BuscarClientesVehiculo()
  Variable RouteTSP: inicializarStruct()
  Variable listaSecuenciaRuta: [∅]
  // Realizar ruta de camión puro
  if el tipo de transporte de la ruta es 0 then
    for cliente en la ruta do
      AgregarClienteMasCercanoASecuencia(SecuenciaRuta, matrizDistancias,clientes)
    end for
    agregar a RouteTSP la secuencia de la ruta, la distancia y tipo de ruta camión puro
    // Realizar ruta de vehículo completa o pura
  else if el tipo de transporte de la ruta es 1 then
    // Crear ruta de vehículo pura
    if todos los clientes son de vehículo then
      for cliente in listaClientesVehiculos do
        AgregarClienteMasCercanoASecuencia(SecuenciaRuta, matrizDistancias,clientes)
      end for
      agregar a RouteTSP la secuencia de la ruta, la distancia y tipo de ruta vehículo pura
      // Crear ruta de vehículo completa
    else if hay clientes en la listaClientesCamión then
      Variable TruckSubtour: inicializarStruct()
      // Crear main tour
      for cliente en listaClientesVehiculo do
        AgregarClienteMasCercano(listaSecuenciaRuta, matrizDistancias,clientes)
      end for
      agregar a RouteTSP la secuencia de la ruta, la distancia y tipo de ruta maintour
      // Crear ruta subtour
      for cliente en listaClientesCamion do
        // Inicializar un subtour
        if es el primer cliente then
          encontrar el cliente del main tour más cercano al cliente de camión
          agregar a TruckSubtour el cliente raiz, el cliente de camión y el tipo de ruta camión
        else
          buscar el cliente de la ruta completa más cercano al cliente de camión
          // Crear un nuevo subtour
          if es un cliente de maintour y el cliente de camión aun no se ha agregado a un
          subtour then
            crear una nueva subruta
            agregar a la subruta el cliente root y el de camión
          else
            // Unir a un subtour ya existente
            agregar el cliente de camión después del cliente del subtour más cercano.
          end if
        end if
      end for
    end if

  end if
end for
=0
```

Luego de obtener la solución inicial, continuamos con la aplicación de la heurística de mejora. En esta oportunidad se utilizó *HC* con mejor mejora. El movimiento seleccionado por conveniencia es **swap** entre dos clientes, ya sea con los que se encuentren en su misma ruta o bien en otra. La técnica es básicamente la que utilizó Chao [3] llamada *Two-point descent exchange*, en donde se excluyen swap de dos casos:

- Mover un cliente de camión a un main tour de una ruta vehículo completa o a una ruta de vehículo pura.
- Mover el cliente raíz de un main tour.

El criterio de término del algoritmo es cuando no hay una mejor solución que la actual o bien se acabó el número de iteraciones. También cabe destacar que utilice un vector de tuplas para representar los movimientos, compuesto de la siguiente forma:

```
vector<tuple<float,int,int,RouteTSP,RouteTSP>> listMovimientos;
```

- Primer valor es la suma total recorrida (grafo completo) al realizar el intercambio de dos clientes, vendría siendo la función de evaluación.
- El segundo valor es un entero que representa el id del primer cliente.
- El tercer valor es un entero que representa el id segundo cliente.
- El cuarto valor es una estructura de tipo RouteTSP que representa la ruta del primer cliente.
- El quinto valor una estructura de tipo RouteTSP que representa la ruta del segundo cliente.

Otra aclaración importante es que existen dos vectores de rutas. La primera, es el vector *listaRoutes* que contiene rutas de vehículo puro, camión puro y main tours. La segunda, es el vector *listaSubtours* contiene solo rutas de subtours.
También cada movimiento se considera factible solo si se siguen cumpliendo las restricciones de las demandas al realizar el swap. Finalmente, el algoritmo verifica la lista de movimientos (vecinos), selecciona el con menor valor de distancia total recorrida y actualiza los nodos en las rutas, en caso que sea menor que la solución actual.

Algorithm 3 Algoritmo Hill Climber

```
while hay un vecino mejor que la solución actual y queden iteraciones disponibles do
  Variable listaMovimientos: [∅]
  Variable mínimoActual: distanciaTotal()
  // Iterar sobre las rutas sin los subtours
  for ruta1 en listaRutas do
    for cliente en ruta1 do
      for ruta2 en listaRutas do
        for cliente2 en ruta2 do
          if tipo de la ruta1 es vehículo completa o puro then
            // Solo podemos intercambiar con clientes que sean de vehículo tipo 0
            if cliente2 es tipo 0 && cliente1 ≠ depósito && cliente2 ≠ depósito then
              if ruta1 ≠ ruta2 then
                // Para que un movimiento sea factible se deben seguir respetando las demandas de las rutas
                if movimientoFactible() then
                  // tupla que contiene el computo del cambio, como la distancia total recorrida de todas las rutas
                  Variable tupla: ComputarCambio (cliente1, cliente2, ruta1, ruta2)
                  Variable sumaTotal: tupla[2]
                  listaMovimientos.push(sumaTotal, cliente, cliente2, ruta, ruta2, 1)
                end if
                // Caso en que se intercambian nodos de la misma ruta y se llama a ComputarCambio modificado con parámetro 2
              else if ruta1 = ruta2 && cliente1 ≠ cliente2 then
                Variable tupla: ComputarCambio (cliente1,cliente2, ruta1, ruta2)
                Variable sumaTotal: tupla[2]
                listaMovimientos.push(sumaTotal, cliente, cliente2, ruta, ruta2, 2)
              end if
            end if
            // En este caso se puede intercambiar con cualquier tipo de cliente, tanto 0 como 1
          else if tipo de ruta1 es camión puro && cliente1 ≠ depósito && cliente2 ≠ depósito
          then
            if ruta1 ≠ ruta2 then
              if movimientoFactible() then
                Variable tupla: ComputarCambio (cliente1, cliente2, ruta1, ruta2)
                Variable sumaTotal: tupla[2]
                listaMovimientos.push(sumaTotal, cliente, cliente2, ruta, ruta2, 1)
              end if
            end if
          else if ruta1 = ruta2 && cliente1 ≠ cliente2 then
            Variable tupla: ComputarCambio (cliente1, cliente2, ruta1, ruta2)
            Variable sumaTotal: tupla[2]
            listaMovimientos.push(sumaTotal, cliente, cliente2, ruta, ruta2, 2)
          end if
        end for
      end for
    end for
  end for
end while =0
```

Algorithm 4 Continuación algoritmo

```
// Iterar sobre los subtours
for ruta1 en listaSubtours do
  for cliente1 en ruta1 do
    for ruta2 en listaSubtours do
      for cliente2 en ruta2 do
        if cliente1 no es raíz && cliente2 no es raíz then
          if ruta1 ≠ ruta2 then
            Variable tupla: ComputarCambio (cliente1, cliente2, ruta1, ruta2)
            Variable sumaTotal: tupla[2]
            listaMovimientos.push(sumaTotal, cliente, cliente2, ruta, ruta2, 2)
          else if ruta1 = ruta2 then
            Variable tupla: ComputarCambio (cliente1, cliente2, ruta1, ruta2)
            Variable sumaTotal: tupla[2]
            listaMovimientos.push(sumaTotal, cliente, cliente2, ruta, ruta2, 1)
          end if
        end if
      end for
      // En este caso se itera sobre listaRoutes esperando a encontrar rutas de camión puro
      con la cual pueda ser intercambiado
      for ruta3 en listRoutes do
        if tipo ruta3 es camión puro then
          if cliente3 ≠ depósito && movimientoFactible() then
            Variable tupla: ComputarCambio (cliente1, cliente2, ruta1, ruta2)
            Variable sumaTotal: tupla[2]
            listaMovimientos.push(sumaTotal, cliente, cliente3, ruta, ruta3, 1)
          end if
        end if
      end for
    end for
  end for
end for
Variable mejorMovimiento: encontrarMejorMovimiento(listaMovimientos)
if distanciaMejorMovimiento < mínimoActual then
  ejecutarMejorMovimiento(mejorMovimiento, listRoutes, Subtours)
else
  terminar el ciclo inicial del while
end if =0
```

7. Experimentos

7.1. Metodología de prueba

Para realizar las pruebas del algoritmo, se utilizaron 12 instancias de pruebas [1], estas son las comúnmente utilizadas en la literatura. La técnica utilizada es *HC* con mejor mejora, esta metahurística contiene dos parámetros, la cantidad máxima de iteraciones y el número de reinicios.
Para esta experimentación se dejó constante el número máximo de iteraciones en 1000, mientras que la cantidad de reinicios fueron variables. La idea fundamental es comparar un algoritmo que encuentra solución inicial Greedy (seleccionando siempre al vecino más cercano), con uno que encuentra una solución inicial aleatoria (esta solución hace referencia al problema de *TSP* y no de asignación, continuando los mismos nodos asignados a la misma ruta) y luego ambos aplican la misma heurística de mejora explicada en la sección anterior. También cabe destacar que el algoritmo Greedy al ser determinista siempre encontrara la misma solución, mientras que el aleatorio será variable. Para este último se utilizaran diferentes cantidades de reinicios, siendo los siguientes: 10, 20, 30, 40. Luego se calcularan los promedios de las soluciones encontradas y se compararan con la encontrada por el algoritmo greedy. El criterio de termino del algoritmo será cuando no se encuentre una mejor solución que la actual o bien se alcance el número máximo de iteraciones, como se aplica tradicionalmente en *HC* con mejor mejora.
Finalmente, se realizara una comparación del tiempo de ejecución para las diferentes instancias del problemas, cuando se utiliza el algoritmo greedy junto a *HC* y por temas de construcción del algoritmo no se puede comparar los tiempos con la solución aleatoria, ya que esta simplemente desordena aleatoriamente la que crea el algoritmo greedy inicialmente.

Instancia	v.c	t.c	total	cantidadC	capacidadC	cantidadT	capacidadT
TTRP_01	38	12	50	5	100	3	100
TTRP_02	25	25	50	5	100	3	100
TTRP_03	13	37	50	5	100	3	100
TTRP_04	57	18	75	9	100	5	100
TTRP_05	38	37	75	9	100	5	100
TTRP_06	19	56	75	9	100	5	100
TTRP_08	50	50	100	8	150	4	100
TTRP_10	113	37	150	12	150	6	100
TTRP_11	75	75	150	12	150	6	100
TTRP_12	38	112	150	12	150	6	100
TTRP_13	150	49	199	17	150	9	100
TTRP_15	50	149	199	17	150	9	100

Cuadro 1: Instancias de prueba

7.2. Entorno de experimentación

Todas las pruebas fueron realizadas en una máquina con las siguientes especificaciones:

- Procesador Intel(R) Core(TM) i5-8300H CPU @ 2.30GHz de 64 bits y 4 núcleos, junto a 16gb de ram.
- Las pruebas fueron realizadas en sistema operativo Windows 11 con subsystem for linux (WSL)

7.3. Comparación con el Estado del Arte

El algoritmo fue aplicado con las mismas instancias del estado del arte, sin embargo, los movimientos utilizados en estos, son bastante más elaborados que el swap realizado en esta experimentación. Por ejemplo, Chao [3] utilizada 3 pasos de mejora, entre ellas, cambiar nodo raíz, la cual no fue implementada en este algoritmo. También una limitante, es la forma en que se utilizaron semillas para crear las rutas y al momento de resolver la asignación, ya que se limita significativamente el espacio de búsqueda.

8. Resultados

Como se puede apreciar en los resultados de la tabla [2], los resultados fueron bastante similares en todos los experimentos. En su mayoría, los experimentos con soluciones iniciales aleatorias generan levemente un valor menor de la función objetivo que en el caso del algoritmo greedy. La mayor diferencia se puede apreciar en el gráfico [9], en donde la mejor solución supero por 49 unidades al algoritmo greedy. Para los otros casos las diferencias no fueron muy significativas, como se pueden apreciar en los gráficos [10] y [11], habiendo una diferencia de unas 20 unidades (no se incluyen más gráficos porque son resultados similares), lo que demuestra que el algoritmo greedy de por sí ya encuentra una solución inicial bastante buena. También se encontro que en 6 de los 12 casos, el mejor promedio de soluciones fue al realizar un re-start 40 veces. Finalmente, hay 2 casos en los que gano el algoritmo greedy.

Instancia	Greedy	re-start-10	re-start-20	re-start-30	re-start-40
TTRP_01	815.389	781.560	782.204	779.374	**766.427**
TTRP_02	848.411	828.234	830.201	823.298	**822.818**
TTRP_03	859.289	**851.522**	853.592	853.096	859.572
TTRP_04	1054.32	**1036.46**	1040.2	1047.272	1041.91475
TTRP_05	1198.37	1189.48	1191.829	1187.638	**1186.40675**
TTRP_06	1295.24	1285.92	**1284.607**	1285.37	1285.86825
TTRP_08	1223.44	1207.804	1214.349	1207.394	**1204.316**
TTRP_10	**1411.16**	1447.031	1438.6905	1452.156667	1429.965
TTRP_11	1636.34	1655.415	**1630.612**	1643.096333	1631.859231
TTRP_12	**1784.07**	1807.083	1798.0615	1796.375	1794.674
TTRP_13	1775.53	1768.471	1773.7765	1772.529667	**1759.63275**
TTRP_15	2256.35	2241.684	2240.05	2244.305333	**2238.63325**

Cuadro 2: Resultado de las instancias de prueba

Por el lado de los tiempos de ejecución [12], mientras mayor cantidad de clientes, camiones y trailers, mayor es el tiempo de ejecución del algoritmo, sin embargo, son tiempos razonables y el único caso que aumenta significativamente el tiempo es en la instancia TTRP_15, ejecutándose por un minuto, lo cual se puede deber a la gran cantidad de clientes de camión.

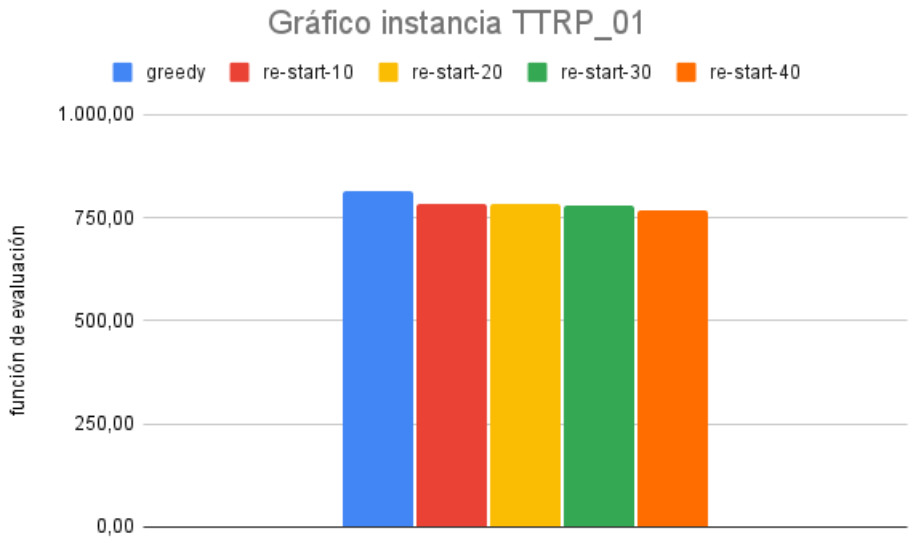


Figura 9: Gráfico comparación entre solución inicial greedy y aleatoria para instancia TTRP_01

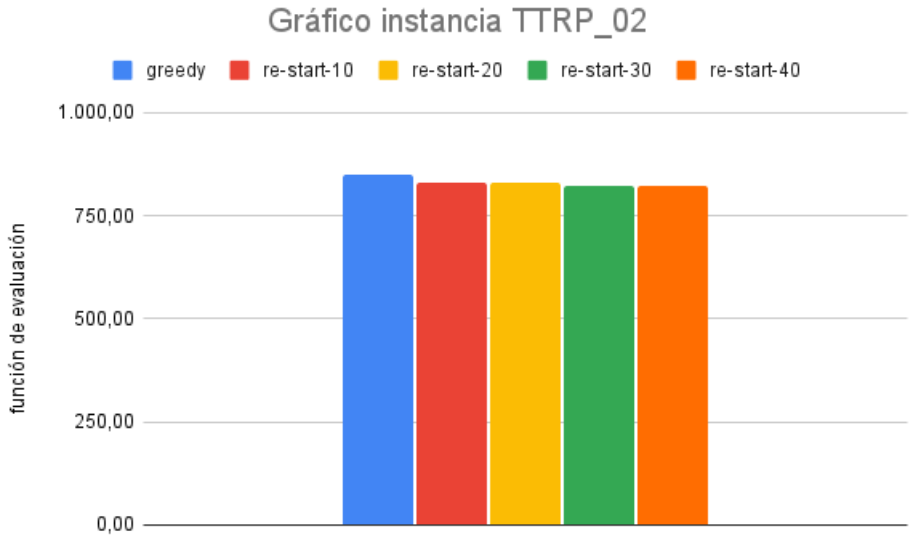


Figura 10: Gráfico comparación entre solución inicial greedy y aleatoria para instancia TTRP_02

9. Conclusiones

TTRP es un problema bastante complejo e importante en las últimas dos décadas. Si bien, a principios de los 2000 hasta el 2013 hubo diversos estudios, en los años posteriores comenzaron a disminuir. En los paper revisados entre el 2002 y 2015, los autores utilizan diversas técnicas para resolver el problema, sin embargo el único planteamiento matemático que logre encontrar fue el de Chao[3]. Los siguientes investigadores plantean enfoques directos utilizando heurísticas de construcción y mejoras, como: *T-Cluster* y *T-Sweep*, *Simulated Annealing* y *Tabu search*.

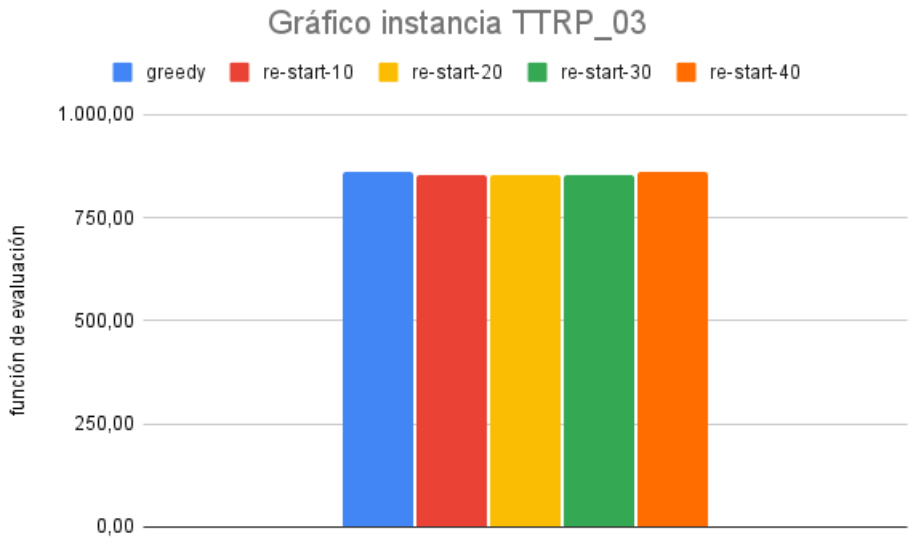


Figura 11: Gráfico comparación entre solución inicial greedy y aleatoria para instancia TTRP_03

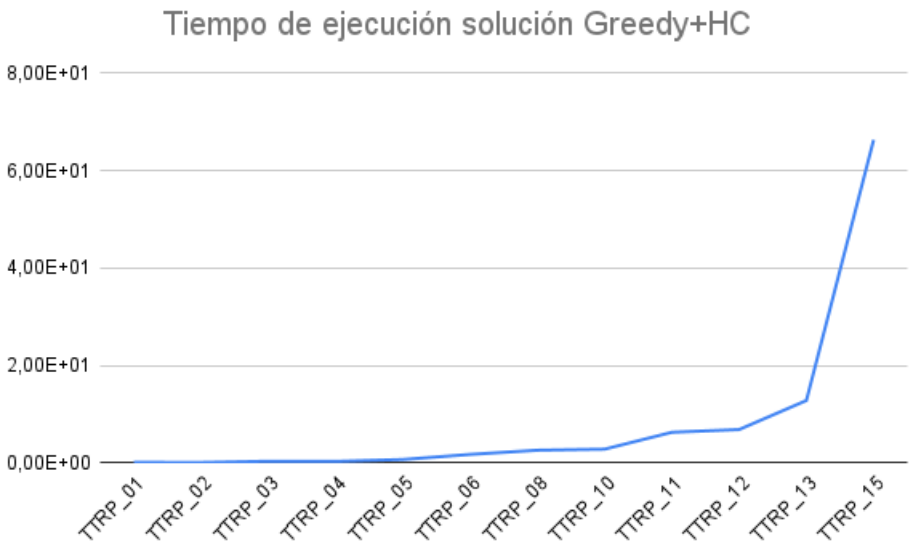


Figura 12: Gráfico de tiempo de ejecución para algoritmo Greedy + HC en segundos

Casi todos los investigadores utilizan la misma idea base, que trata de buscar formas de intercambiar las rutas y hacer modificaciones al grafo utilizando métodos de búsqueda local, sin embargo lo que difiere es el procedimiento, ya que algunos utilizan conceptos de aleatoriedad, otros restricciones tabús o algoritmos codiciosos.

9.1. Análisis de los resultados

Se llegó a la conclusión que encontrar una solución inicial aleatoria para el problema TSP y luego aplicar la metahurística de mejorar *HC*, logra mejores soluciones que aplicar una construcción greedy y luego *HC*. Sin embargo, greedy requiere una menor cantidad de iteraciones cuando utiliza la heuristíca de mejora y llega a soluciones suficientemente buenas, habiendo una discrepancia no significativa. En el mejor de los casos se reportó una diferencia de 49 unidades y en otros la diferencia rondaba entre las 10 y 20 unidades.
Los tiempos de ejecución del algoritmo no son excesivos, incluso el único valor que aumentó significativamente es la instancia TTRP_15, en la cual se demora 1 minuto y se cree que es por la cantidad de clientes (particularmente una gran cantidad de clientes camión puro) y camiones. Finalmente en 6 de los 12 casos, se encontraron mejor promedio de soluciones al aplicar 40 restarts

9.2. Análisis de la tecnica utilizada

El uso de *HC* con mejor mejora encontró soluciones factibles y de buena calidad, a pesar de las limitantes que contiene (la falta de exploración en comparación a otras metahurísticas), también se puede mencionar que es una técnica costosa computacionalmente, porque requiere calcular todo el espacio de búsqueda antes de realizar un movimiento, sin embargo, a pesar de esto los tiempos siempre fueron razonables.

9.3. Las fallas

El problema mayor es que se limitó fuertemente el espacio de búsqueda al utilizar semillas para cada ruta, lo que no permitió explorar otras soluciones y hubiese sido bueno experimentar con soluciones aleatorias al momento de resolver el problema de asignación. También cabe destacar que Chao [3] menciona que la mejor solución fue encontrada al elegir como primera semilla la décima más lejana del depósito, sin embargo, en este experimento solo se utilizó la más lejana del depósito como primera semilla. Otra falla encontrada fue el hecho de crear directamente un algoritmo greedy que resolviera el problema de TSP, construyendo las rutas con una función miope que uniera al vecino más cercano, esto podría hacer que *HC* se estancara rápidamente y de hecho lo hacía (aunque encontraba soluciones aceptables), realizando muy pocas iteraciones, es por esto mismo que se intentó corregir el error aplicando una función que desordenara todas las rutas aleatoriamente y en cada re-start se creaba una secuencia TSP totalmente diferente.

9.4. Posibles mejoras y trabajo futuro

Las posibles mejoras a realizar y trabajo futuro, es probar con otras semillas iniciales como lo hizo Chao, también no resolver el problema de asignación en forma tan determinista, sino que utilizar más aleatoriedad y de esta forma poder explorar más. Otra mejora fundamental es definir nuevos movimientos, como cambiar los nodos raíces, lo cual ayuda a encontrar nuevas sub rutas con menor distancia. Finalmente, se debería probar con otras metahurísticas de mejora, ya que *HC* es limitado y no explora adecuadamente (solo con restarts) en comparación a otras técnicas.

10. Bibliografía

Referencias

[1] Gillett BE. A heuristic algorithm for the vehicle-dispatch problem. *European Journal of Operational Research*, 22(2):340–9, 1974.

[2] Ulrich D. Truck and trailer routing—problems, heuristics and computational experience. *Computers and Operations Research*, 40(2):536–546, 2013.

[3] Chao I.M. A tabu search method for the truck and trailer routing problem. *Computers and Operations Research*, 1(29):33–51, 2002.

[4] Gerdessen JC. Vehicle routingproblem with trailers. *European Journal of Operational Research*, 93(1):135–47, 1996.

[5] Villegas JG. A grasp with evolutionary path relinking for the truck and trailer routing problem. *Computers and Operations Research*, 38(1):1319–34, 2011.

[6] Lin S-W. Solving the truck and trailer routing problem based on a simulated annealing heuristic. *Computers and Operations Research*, 36(1):1963–92, 2009.

[7] Scheurer S.A. A tabu search method for the truck and trailer routing problem. *Computers and Operations Research*, 1(33):894–909, 2006.

[8] Isis T. International journal of computational intelligence systems. *Solving the Truck and Trailer Routing Problem with Fuzzy Constraints*, 8(4):713–724, 2015.