Un Algoritmo para Empaquetado, Secuenciación y Operaciones de Recogida en un Almacén

Borja Menéndez, Manuel Bustillo, Eduardo G. Pardo y Abraham Duarte

Dept. Informática y Estadística, Universidad Rey Juan Carlos,
Móstoles, Madrid, España
borja.menendez@urjc.es
m.bustilloa@alumnos.urjc.es
{eduardo.pardo,abraham.duarte}@urjc.es
http://grafo.etsii.urjc.es/

Resumen El empaquetado de pedidos es un problema de optimización relacionado con el proceso de recogida de pedidos en un almacén. Consiste en agrupar pedidos recibidos en un almacén (cada pedido está compuesto por una lista de productos que recoger) en un conjunto de lotes con una capacidad máxima fija. Después se ordenan los lotes y se forman las rutas para recoger los productos del mismo lote. En este artículo se aborda el Problema de Empaquetado y Secuenciación de Pedidos en el que cada pedido tiene una hora límite de entrega. Este problema consiste en agrupar pedidos en lotes y ordenarlos de tal forma que el retraso de cada pedido (el tiempo extra sobre la hora límite) sea mínimo. En este artículo se propone un algoritmo basado en la metodología Búsqueda de Vecindad Variable para abordar el problema. La aproximación propuesta mejora métodos previos del estado del arte.

Keywords: Búsqueda de Vecindad Variable, GVNS, Problema de Empaquetado y Secuenciación de Pedidos

1. Introducción

Los sistemas de gestión de almacenes se han convertido en parte esencial de la estrategia de la cadena de suministro. Se centran principalmente en mover y almacenar materiales dentro de un almacén realizando diferentes transacciones (incluyendo envío, recepción y recogida). Se han propuesto varias políticas en la literatura dirigidas a la mejora de las operaciones en almacenes. Algunas de ellas se pueden modelar como problemas de optimización [10]. A nivel estratégico, una de las decisiones más importantes es el diseño del almacén [4], puesto que tiene un gran impacto en la longitud del recorrido de recogida en un almacén [6]. A pesar de ello, el éxito de la gestión de almacenes reside, en gran medida, en la manera en la que se recogen los pedidos de los clientes. El proceso de recogida consiste en recolectar una cantidad de productos antes del envío. Este proceso puede consumir hasta un 60 % del tiempo total de las actividades de un almacén, lo que puede suponer más de la mitad de los costes totales de operación [3].

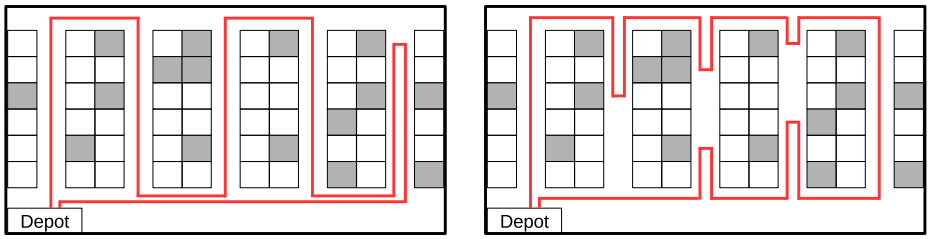
Dado un conjunto de pedidos recibidos en un almacén, hay dos estrategias de recogida de pedidos básicas: *recogida por orden estricto* y *empaquetado*. En la primera, los trabajadores recogen los pedidos a medida que van llegando. En la segunda, varios pedidos se ponen juntos en un mismo lote, siempre que satisfagan una restricción de capacidad. Después se asigna cada lote a un trabajador, que puede recoger los productos de cualquier pedido agrupado en ese mismo lote.

Reducir el tiempo de viaje de las operaciones de recogida mejora la productividad. Por ello se usan estrategias de empaquetado en almacenes. De acuerdo con [2], si el empaquetado y el enrutamiento se consideran de manera simultánea, los beneficios asociados se incrementan considerablemente, reduciendo los tiempos de viaje más de un 35 %. Aunque este artículo se centra en las estrategias de empaquetado y secuenciación para minimizar el retraso total, la función objetivo se evalúa a través de un algoritmo de enrutamiento.

El enrutamiento determina la secuencia de pasos para recoger todos los productos del mismo lote. Asimismo, determina el tiempo necesario para recoger los pedidos. El problema del enrutamiento se puede resolver de manera exacta en tiempo polinomial [12]. No obstante, las rutas óptimas pueden ser ilógicas para los trabajadores humanos. Por tanto, este tipo de soluciones se suelen descartar en situaciones reales, recurriendo a estrategias heurísticas.

En la Figura 1a se muestra un ejemplo de almacén con 5 pasillos paralelos del mismo tamaño, dos pasillos transversales (uno frontal y otro trasero), 10 estanterías con 6 ubicaciones de productos cada una y un depósito de productos en el pasillo frontal (Depot).

La estrategia heurística de enrutamiento más sencilla, ilustrada con una línea roja en la Figura 1a, es la denominada S-Shape (en forma de S) [5]. Consiste en atravesar por completo todos aquellos pasillos que contienen al menos un producto que recoger. Si el número de pasillos es impar, el último pasillo solo se atraviesa hasta el producto más alejado del pasillo frontal.



(a) Ruta obtenida con S-Shape. (b) Ruta obtenida con Largest Gap.

Figura 1: S-Shape y Largest Gap para la misma configuración de productos.

En la Figura 1b, se muestra la ruta obtenida por la estrategia Largest Gap [1]. En esta estrategia, se recorren enteros tanto el primer como el último pasillo que contengan productos a recoger. El resto de pasillos se recorren sin atravesar el denominado *largest gap*. Este se define como la máxima distancia entre dos

productos consecutivos dentro del mismo pasillo, o la distancia entre el pasillo transversal y el producto más cercano en el pasillo paralelo.

Este artículo se ocupa del Problema de Empaquetado y Secuenciación de Pedidos (OBSP, del inglés *Order Batching and Sequencing Problem*). En el OBSP los pedidos tienen una hora límite de entrega. En el caso de que los pedidos se recojan tras esa hora límite, llevan asociada una penalización. El OBSP consiste en minimizar la suma de los retrasos en la recogida de todos los pedidos recibidos en un almacén. Un pedido está compuesto por varios productos, y estos productos están ubicados en estanterías específicas dentro del almacén. Un conjunto de pedidos agrupados juntos forman un lote, que tiene una capacidad máxima. Desde el punto de vista de la optimización se pueden identificar tres problemas diferentes: cómo agrupar pedidos en lotes (empaquetado), cómo determinar qué lote recoger en cada momento (secuenciación) y cómo designar las rutas correspondientes para recogerlos (enrutamiento). También es importante hacer notar que cada pedido debe ser recogido por un único trabajador.

Recientemente se ha estudiado el OBSP en [9] usando una versión modificada de los algoritmos ILS (del inglés *Iterated Local Search*) y ABHC (del inglés *Attribute-Based Hill Climber*) propuestos en [7] y [8], respectivamente. En concreto, el algoritmo ILS realiza iterativamente una fase de perturbación de la solución, para después aplicar una búsqueda local. El algoritmo termina su ejecución tras una serie de iteraciones determinada en [9]. En cambio, el algoritmo ABHC se basa en un principio simple de Búsqueda Tabú. En él, se definen atributos que tienen que mejorar en cada paso de la búsqueda local. El algoritmo termina su ejecución cuando no se mejora ningún atributo.

En este artículo se propone un algoritmo basado en la metodología VNS para abordar el OBSP. En concreto, se centra la atención en las estrategias de empaquetado y secuenciación, haciendo uso de los algoritmos de enrutamiento previamente definidos. El resto del artículo está organizado como sigue. En la Sección 2 se presenta un algoritmo VNS para abordar tanto el empaquetado como la secuenciación. Finalmente, se presentan experimentos y sus conclusiones asociadas en las secciones 3 y 4, respectivamente.

2. Algoritmo de Empaquetado y Secuenciación

La Búsqueda de Vecindad Variable (VNS) [11] es una metaheurística basada en cambios sistemáticos de estructuras de vecindad, con el objetivo es alcanzar diferentes óptimos locales. El esquema original se ha evolucionado ampliamente con muchas extensiones. En este trabajo se ha escogido la variante General VNS (GVNS) para abordar el problema.

El algoritmo GVNS empieza inicializando el parámetro k, que define la vecindad a explorar, a la primera vecindad (paso 3). Posteriormente, entra en el bucle principal (pasos 5, 6 y 7). En este bucle, la solución actual, s, se modifica en el procedimiento de perturbación aplicando un movimiento aleatorio en la vecindad k-ésima. Después, la solución obtenida, s', se mejora dentro de un procedimiento VND que considera dos vecindades diferentes. Se utiliza una implementación

estándar para el método de cambio de vecindad (paso 7), que determina cuál es la siguiente vecindad a explorar. En concreto, si la solución s'' mejora a s, el método GVNS considera s'' como la nueva mejor solución ($s \leftarrow s''$) y reinicia la búsqueda desde la primera vecindad ($k = 1$). En cualquier otro caso no se actualiza la solución actual, pero se explora una nueva vecindad ($k = k+1$). Estos tres pasos se repiten hasta que se explore la mayor de las vecindades (k_{max}) sin haber encontrado una mejora. Si no se ha superado el tiempo de ejecución máximo (t_{max}), el GVNS realiza una nueva iteración.

Algoritmo 1 Algoritmo GVNS

```
1: procedure GVNS(s, k_max, t_max)
2:     repeat
3:         k ← 1
4:         repeat
5:             s' ← Perturbacion(s, k)
6:             s'' ← VND(s')
7:             CambioDeVecindad(s, s'', k)
8:         until k > k_max
9:         t ← CPUTime()
10:    until t ≥ t_max
11:    return s
12: end procedure
```

2.1. Método Constructivo

El método GVNS parte de una solución factible. Esta solución puede ser generada aleatoriamente, aunque un procedimiento constructivo un poco más elaborado puede mejorar considerablemente la calidad de la mejor solución encontrada. Uno de los métodos constructivos más conocidos para este problema es el conocido como EDD (del inglés *Earliest Due Date*). Este algoritmo constructivo primero ordena todos los pedidos de acuerdo a su hora límite de entrega, de manera que los pedidos con una hora límite más cercana se recogen primero. Después, los pedidos son asignados sucesivamente a lotes siguiendo la organización previamente definida, asegurando en todo caso que no se excede el límite de capacidad. En este sentido, el primer lote se llena hasta que el siguiente pedido que llegue no entre en él. Después se crea un nuevo lote al que se le asigna el pedido anterior, y así sucesivamente. Este método constructivo será utilizado para generar las soluciones de partida del algoritmo GVNS.

2.2. Estructuras de Vecindad

Se han diseñado dos estructuras de vecindad diferentes para este problema. La primera de ellas se denomina N_1 y está basada en movimientos de inserción,

Insert. En ella, un pedido O_i se elimina de su lote actual, B_j, y se añade a otro (B_k) en el que el peso total de sus pedidos, más el del pedido nuevo, no exceda el límite de capacidad del carrito. Considerando estos aspectos, la vecindad N_1 se puede definir más formalmente como:

$$N_1(s) = \{s' \leftarrow Insert(s, O_i, B_j, B_k)\}$$

donde $O_i \in B_j$; $1 \leq i \leq n$, siendo n el número de pedidos; $1 \leq j \leq m, 1 \leq k \leq m$, y $j \neq k$, siendo m el número de paquetes.

La segunda estructura de vecindad es la denominada N_2, basada en intercambios de pedidos (movimientos *Swap*). En este caso, dos pedidos (O_i y O_j) de diferentes lotes (B_k y B_l, respectivamente) se intercambian, de manera que el primer pedido (O_i) se elimina de su lote actual (B_k) para ser añadido al lote B_l y, análogamente, O_j se elimina de B_l y se inserta en B_k. Destacar que este movimiento tiene que ser factible también. Esto quiere decir que solo se puede realizar este movimiento si el peso de cada lote no supera la capacidad máxima del carrito tras el movimiento. Así, la vecindad N_2 se puede definir más formalmente como:

$$N_2(s) = \{s' \leftarrow Swap(s, O_i, B_j, O_k, B_l)\}$$

donde $O_i \in B_j$, $O_k \in B_l$; $1 \leq i \leq n$, $1 \leq k \leq n$, siendo n el número de pedidos; $1 \leq j \leq m$, $1 \leq l \leq m$, y $j \neq k$, siendo m el número de paquetes.

2.3. Estrategia de Perturbación

La fase de perturbación se introduce normalmente en VNS como una estrategia efectiva para escapar de un óptimo local. Dada una solución s, el procedimiento de perturbación genera aleatoriamente una nueva solución s' en la vecindad k-ésima (denotada como $N_k(s)$) aplicando k movimientos. Concretamente, $N_k(s)$ contiene el conjunto de soluciones que pueden ser alcanzadas aplicando k movimientos consecutivos.

En el OBSP, el movimiento aplicado para la estrategia de perturbación está basado en movimientos de intercambio (*Swap*). En concreto, el procedimiento de perturbación aplica un *Swap* aleatorio k veces consecutivas.

2.4. Métodos de Mejora

En este artículo se propone un método VND como el proceso de búsqueda local dentro de GVNS. Este VND explora las dos vecindades anteriormente descritas con una estrategia de primera mejora (*first improvement*). En ambas vecindades, N_1 y N_2, solo se consideran los movimientos factibles que mejoran el valor de la función objetivo. Dada la vecindad N_1, se define el procedimiento de búsqueda local LS1 como aquel que explora N_1. Esta búsqueda local empieza desde un punto aleatorio y realiza movimientos *Insert* hasta que no se encuentren más movimientos de mejora. De manera similar, se define LS2 como el procedimiento de búsqueda local que explora N_2. De nuevo, esta búsqueda

local empieza en un punto aleatorio, pero en este caso se realizan movimientos *Swap* hasta que se alcance el óptimo local.

Una vez que las dos vecindades previas se han definido, se propone una combinación de ambas en un método VND. El método recibe como parámetros de entrada la solución inicial s y el valor de k_{max} (en este caso, $k_{max} = 2$). Las vecindades se ordenan de manera que se explora primero N_1 y después se explora N_2. Como es común en la comunidad VNS, las vecindades se exploran desde la más pequeña y rápida de explorar hasta la más grande. En este caso, N_1 (basada en movimientos de inserción) es normalmente más pequeña que N_2 (basada en movimientos de intercambio), dado que el número de lotes es normalmente más pequeño que el número de pedidos, el orden dentro del lote no se tiene en cuenta y, además, solo se consideran los movimientos factibles. Este método termina cuando no hay mejoras en ninguna de las vecindades.

3. Experimentación

En esta sección se presentan los experimentos realizados para estudiar empíricamente la influencia de las estrategias propuestas. Posteriormente, se comparará la mejor variante con el mejor algoritmo previo en el estado del arte [9]. Tanto para la experimentación preliminar como para la final, los resultados se comparan con el algoritmo EDD, puesto que es el método de comparación en el estado del arte. Los algoritmos se han implementado en Java 7 y la experimentación se ha realizado en un AMD Phenom II X6 1050T con 2.8 GHz y 4 GB de RAM con un sistema operativo Ubuntu 14.04 de 64 bits. La sección incluye también una descripción detallada de las instancias usadas en la experimentación.

3.1. Instancias

Se ha considerado un conjunto de instancias usadas previamente en la literatura para este problema de optimización. En concreto, se ha seleccionado el `Conjunto HS` [9]. Este conjunto contiene 4800 instancias cuyas principales características se describen a continuación.

El almacén sobre el que se trabaja tiene 900 ubicaciones de almacenamiento, donde cada ubicación guarda un producto diferente. Está compuesto por 10 pasillos con 90 ubicaciones cada uno (45 a cada lado del pasillo). El tamaño de cada ubicación es de 1 unidad de longitud (UL), mientras que su anchura es de 1.5 UL. Cuando el trabajador sale de un pasillo paralelo, se asume que se mueve 1 UL hasta el pasillo transversal. Finalmente, el trabajador se mueve 5 UL desde el pasillo actual al inmediatamente posterior. El depósito está situado a 1.5 UL de la primera ubicación del pasillo más a la izquierda.

Existen dos escenarios diferentes: (1) distribución ABC y (2) distribución aleatoria. En la primera los productos se pueden agrupar en tres clases. La Clase A, que contiene productos muy demandados, donde el 10 % de los artículos representan el 52 % de la demanda (los pedidos se guardan en el primer pasillo). La clase B, con productos de demanda media, donde 30 % de los artículos

acapara el 36 % de la demanda (siguientes tres pasillos). Por último, la Clase C, que contiene productos con una baja demanda y representan el 60 % restante de los productos, aunque su demanda solo es el 12 % del total (resto de pasillos). En cualquier caso, los productos se organizan de manera aleatoria dentro de una misma clase. En el segundo escenario, los productos se distribuyen aleatoriamente entre todas las ubicaciones.

Este conjunto de instancias considera 4 tamaños diferentes de pedidos ($n = \{20, 40, 60, 80\}$), donde el número de productos por pedido se distribuye uniformemente en $\{5, 6, \ldots, 25\}$. Adicionalmente, se han considerado dos capacidades diferentes para el carrito, 45 y 75 productos por lote.

Para determinar el momento en el que se recoge un producto, es necesario conocer la velocidad a la que circula un trabajador. En este caso, la velocidad es de 0.48 UL/segundo. El tiempo que tarda en recoger cada producto de su ubicación es 6 segundos. También existe un tiempo de configuración del lote de 3 minutos, sin importar la cantidad de pedidos en el paquete. Estos parámetros se establecen de acuerdo a la configuración de las instancias.

Por último, en este conjunto de instancias considera también el parámetro de MTCR (del inglés *Modified Traffic Congestion Rate*). Valores altos en el MTCR hacen que las fechas de vencimiento de los pedidos estén muy cerca entre sí, mientras que valores bajos del MTCR indican que estas fechas de vencimiento estén más dispersas. Se han considerado valores de MTCR iguales a 0.50, 0.55, 0.60, 0.65, 0.70 y 0.75.

Se ha probado experimentalmente que no es necesario usar el conjunto total de instancias, ya que un subconjunto pequeño representativo puede producir resultados similares. Así pues, se ha seleccionado un subconjunto de 96 instancias del `Conjunto HS` mencionado con anterioridad. Concretamente, para realizar los experimentos se ha seleccionado una instancia por cada tipo.

3.2. Experimentos preliminares

En esta sección se ajustan los parámetros del algoritmo (valor de k_{max}, búsqueda local a utilizar y selección del algoritmo de enrutamiento) realizando varios experimentos preliminares con 12 de las 96 instancias seleccionadas.

Primero se compara el rendimiento del VND respecto a los procedimientos de búsqueda local aisladamente (LS1 y LS2). Para realizar esta comparación se ha construido una solución con el método EDD y después se ha aplicado cada uno de los métodos de mejora (LS1, LS2 y VND) a la misma solución inicial.

En las Tablas 1 y 2 se muestra, para cada método de mejora, el valor medio de la función objetivo (*Tardiness*), el valor medio del tiempo de ejecución (*CPU t (s)*), la media de la desviación respecto a la solución EDD (*Dev. (%)*), el número de veces que el método ha obtenido la mejor solución (*#Best*) y el número de veces que el método ha obtenido un valor de retraso igual a 0 (*#Zeros*). En la Tabla 1, se presentan los resultados de cada algoritmo con la estrategia de enrutamiento S-Shape y en la Tabla 2, los obtenidos con Largest Gap. Los resultados en las tablas confirman que el procedimiento VND obtiene mejores resultados respecto a los métodos de búsqueda local cuando se usa tanto la

estrategia S-Shape como Largest Gap, ya que el método VND obtiene la máxima desviación respecto a EDD. Es importante destacar que cuanto mayor es la desviación con respecto al método de partida, mayor es la mejora. También obtiene el menor *Tardiness* y el mayor número de #Best del experimento, tanto para S-Shape como para Largest Gap.

Mejora	**LS1**	**LS2**	**VND**
Tardiness	28462	20665	16603
CPU t (s)	<1	<1	<1
Dev. (%)	21.13 %	31.76 %	38.75 %
#Best	3	4	11
#Zeros	1	1	1

Tabla 1: Comparativa de LS1, LS2 y VND con S-Shape.

Mejora	**LS1**	**LS2**	**VND**
Tardiness	17012	17418	12040
CPU t (s)	<1	<1	<1
Dev. (%)	14.10 %	17.96 %	26.68 %
#Best	3	4	9
#Zeros	1	1	1

Tabla 2: Comparativa de LS1, LS2 y VND con Largest Gap.

El siguiente experimento consiste en determinar la influencia del parámetro k_{max} en el rendimiento del GVNS, fijando el procedimiento de búsqueda local como el método VND. El tiempo de ejecución en el siguiente experimento depende del número de pedidos (100 ms × número de pedidos en la instancia) y los resultados se comparan respecto al método EDD. En particular, se considera $k_{max} = \{10, 15, 20\}$. En la Tabla 3 se muestran los resultados asociados a cada estrategia de enrutamiento. Se presentan las mismas filas que en la Tabla 1. Para cada estrategia de enrutamiento el mejor valor de k_{max} es 15, ya que obtiene el mejor valor para la desviación respecto al EDD y el mayor número de mejores soluciones encontradas.

Enrutamiento	**S-Shape**			**Largest Gap**		
$\mathbf{k_{max}}$	**10**	**15**	**20**	**10**	**15**	**20**
Tardiness	11590	11383	11672	9895	10032	10116
CPU t (s)	3.36	3.37	3.42	3.30	3.32	3.36
Dev. (%)	53.72 %	53.82 %	53.70 %	47.99 %	49.43 %	49.36 %
#Best	9	11	9	10	10	10
#Zeros	1	1	1	2	3	3

Tabla 3: Influencia de k_{max} en el rendimiento del GVNS.

3.3. Experimentos Finales

Una vez se han identificado los mejores parámetros y estrategias entre las variantes propuestas, la experimentación final está dedicada a comparar el rendimiento de la mejor propuesta con el mejor método del estado del arte. En

concreto, se ha seleccionado VND como estrategia de mejora y $k_{max} = 15$ para configurar el algoritmo. Una vez configurado, se ha comparado con el algoritmo ILS descrito en [9] en ambos contextos: para la estrategia S-Shape y para la estrategia Largest Gap.

Para la comparativa final se ha tomado el conjunto seleccionado de 96 instancias. Los algoritmos se ejecutaron durante el mismo tiempo, que se estableció basándose en el tiempo promedio por instancia que necesitó el algoritmo ILS con un máximo de 300 segundos por instancia.

En las Tablas 4 y 5 se reportan los resultados obtenidos por el algoritmo ILS comparados contra los obtenidos por el GVNS. En concreto, en la Tabla 4 se muestran los resultados obtenidos usando S-Shape como estrategia de enrutamiento. Análogamente, en la Tabla 5 se muestran los resultados usando la estrategia Largest Gap.

En la Tabla 4 se puede ver que ILS obtiene mejores resultados en términos de desviación respecto al EDD, sin embargo, es el GVNS propuesto el que obtiene un mayor número de mejores soluciones. Para confirmar dichas observaciones se ha realizado un test de Wilcoxon. El p-valor obtenido es de 0.981, que indica que no existen diferencias significativas entre ambos métodos.

Algoritmo	**ILS**	**GVNS**
Tardiness	15654	15697
CPU t (s)	9.26	8.60
Dev. (%)	53.50 %	53.13 %
#Best	74	75
#Zeros	11	11

Tabla 4: Mejores métodos del `Conjunto HS` con S-Shape.

Algoritmo	**ILS**	**GVNS**
Tardiness	13952	13622
CPU t (s)	111.21	97.31
Dev. (%)	49.44 %	50.00 %
#Best	69	86
#Zeros	14	14

Tabla 5: Mejores métodos del `Conjunto HS` con Largest Gap.

En cambio, en la Tabla 5 parece que las diferencias entre ambos métodos son mayores. La desviación respecto al EDD obtenida por el GVNS es mayor que la obtenida por el ILS. Además, el GVNS es capaz de obtener más mejores soluciones que el ILS. De nuevo, para confirmar esta observación se ha realizado un test de Wilcoxon. El p-valor obtenido de 0.001 indica que existen diferencias significativas entre los algoritmos.

En las Tablas 6 y 7 se presentan los resultados divididos por grupos, dependiendo del parámetro MTCR. En concreto, se reporta la media de desviación con respecto al método EDD para cada algoritmo usando la estrategia de enrutamiento S-Shape (Tabla 6) y Largest Gap (Tabla 7). Desde esta perspectiva es interesante señalar que el rendimiento del ILS y el GVNS respecto al EDD es mayor cuando el MTCR es igual a 0.60 en ambos casos. También en ambos casos, cuando el MTCR es mayor que 0.60 el rendimiento es mayor que en los casos en los que el MTCR es menor que 0.60.

A pesar de los resultados mostrados en las Tablas 4 y 5, es justo decir que la estrategia de enrutamiento Largest Gap es la mejor para este conjunto de

MTCR	ILS	GVNS
0.50	38.36 %	35.95 %
0.55	36.27 %	36.48 %
0.60	66.85 %	67.30 %
0.65	60.03 %	60.09 %
0.70	57.37 %	57.31 %
0.75	62.14 %	61.83 %

Tabla 6: Desviación media respecto al EDD considerando S-Shape.

MTCR	ILS	GVNS
0.50	38.96 %	39.08 %
0.55	33.38 %	34.67 %
0.60	60.97 %	61.21 %
0.65	57.52 %	57.65 %
0.70	51.34 %	52.39 %
0.75	54.46 %	55.00 %

Tabla 7: Desviación media respecto al EDD considerando Largest Gap.

instancias. Esto se muestra en la Tabla 8, donde se mezclan los datos de la Tabla 4 y la Tabla 5. En esta tabla se comparan los cuatro algoritmos previos (ILS+SS y GVNS+SS para los algoritmos que utilizan el método de enrutamiento S-Shape, ILS+LG y GVNS+LG para los algoritmos que utilizan el método Largest Gap). En este caso no se reporta la desviación respecto al EDD puesto que el punto de comparación es diferente para los casos de S-Shape y Largest Gap. Sin embargo, en lugar de ese valor se ha reportado la desviación respecto a la mejor solución encontrada (Dev. Best (%)) por cualquiera de los métodos (hay que señalar que, en este caso, cuanto mayor es la desviación, peor es el algoritmo). Cuando el valor de la función objetivo es cero, la desviación no se puede obtener, por lo que también se reporta el número de veces que se obtiene ese valor (#Zeros). Los resultados en la Tabla 8 ayudan a confirmar que el algoritmo propuesto (GVNS) usando la estrategia de enrutamiento Largest Gap es la mejor estrategia. En concreto es el que obtiene la menor desviación respecto al mejor valor del experimento (1.25 %) y el mayor número de mejores soluciones encontradas (79 de 96 instancias).

Algoritmo	ILS+SS	GVNS+SS	ILS+LG	GVNS+LG
Tardiness	15654	15697	13952	13622
Dev. Best (%)	44.75	56.35	3.40	1.25
#Best	18	16	64	79
#Zeros	11	11	14	14

Tabla 8: Comparativa de los mejores cuatro métodos considerando la desviación respecto de la mejor solución encontrada.

4. Conclusiones

En este artículo se presenta un algoritmo General VNS para abordar el Problema de Empaquetado y Secuenciación de Pedidos (OBSP). Concretamente se hace uso del conocido método constructivo (EDD). GVNS explora dos vecindades diferentes emparejadas en un método VND. El algoritmo obtenido se ha

unido con dos métodos diferentes de enrutamiento: S-Shape y Largest Gap. Después de probar diferentes configuraciones para el GVNS, se ha comparado la mejor con el mejor algoritmo previo del estado del arte sobre un conjunto de instancias referenciadas en la literatura. Esta comparativa, apoyada estadísticamente, favorece al método GVNS propuesto.

5. Agradecimientos

Este trabajo ha sido parcialmente financiado por el Ministerio de Economía y Competitividad de España, a través del proyecto con referencia TIN2012-35632-C02-02, y la Comunidad de Madrid, a través del proyecto con referencia S2013/ICE-2894.

Referencias

[1] De Koster, M.B.M., Van der Poort, E.S., Wolters, M.: Efficient order batching methods in warehouses. International Journal of Production Research. 37 (7), 1479–1504 (1999)
[2] De Koster, R., Roodbergen, K.J., Van Voorden, R.: Reduction of walking time in the distribution center of De Bijenkorf. New trends in distribution logistics. Pages 215–234, Springer (1999)
[3] Drury, J.: Towards more efficient order picking. IMM monograph. 1 (1988)
[4] Ghiani, G., Laporte, G., Musmanno, R.: Introduction to logistics systems planning and control. John Wiley & Sons (2004)
[5] Goetschalckx, M., Ratliff, H.D.: Order Picking In An Aisle. IIE Transactions. 20 (1), 53–62 (1988)
[6] Gu, J., Goetschalckx, M., McGinnis, L.F.: Research on warehouse operation: A comprehensive review. European Journal of Operational Research. 177 (1), 1–21 (2007)
[7] Henn, S., Koch, S., Doerner, K. F., Strauss, C., Wäscher, G.: Metaheuristics for the order batching problem in manual order picking systems. BuR-Business Research. 3(1), 82-105 (2010)
[8] Henn, S., Wäscher, G.: Tabu search heuristics for the order batching problem in manual order picking systems. European Journal of Operational Research. 222(3), 484-494 (2012)
[9] Henn, S., Schmid, V.: Metaheuristics for order batching and sequencing in manual order picking systems. Computers & Industrial Engineering. 66 (2), 338–351 (2013)
[10] Karasek, J.: An overview of warehouse optimization. Computers in Industry. 2 (3), 111–117 (2013)
[11] Mladenović, N., Hansen, P.: Variable neighborhood search. Computers & Operations Research. 24 (11), 1097–1100 (1997)
[12] Ratliff, H.D., Rosenthal, A.S.: Order-picking in a rectangular warehouse: a solvable case of the traveling salesman problem. Operations Research. 31 (3), 507–521 (1983)