Journal of Global Optimization https://doi.org/
10.1007/s10898-020-00910-2

GRASP com Variable Neighbourhood Descent para o problema de lote de pedidos online

Sergio Gil-Borrás[1] · Eduardo G. Pardo[2] · Antonio Alonso-Ayuso[2] · Abraham Duarte[2]

Recebido: 24 de maio de 2019 / Aceito: 16 de abril de 2020

Abstrato
O Online Order Batching Problem (OOBP) é uma variante do conhecido Order Batching Problem (OBP). Assim como no OBP, o objetivo desse problema é coletar todos os pedidos que chegam a um depósito, seguindo uma política de separação por lote de pedidos, enquanto minimiza uma função objetivo particular. Portanto, os pedidos são agrupados em lotes, de capacidade máxima pré-definida, antes de serem coletados. Cada lote é atribuído a um único selecionador, que coleta todos os pedidos do lote em uma única rota. Ao contrário do OBP, esta variante apresenta a particularidade de as encomendas consideradas em cada instância não estarem totalmente disponíveis no armazém no início do dia, mas podem chegar ao sistema depois de iniciado o processo de picking. Em seguida, os lotes devem ser atualizados dinamicamente e, como consequência, as rotas também. Nesse artigo, o tempo máximo de giro (tempo máximo que um pedido permanece no armazém) e o tempo máximo de conclusão (tempo total de coleta de todos os pedidos recebidos no armazém) são minimizados. Para tanto, propomos um algoritmo baseado na combinação de um Procedimento de Busca Adaptativa Randomizada Greedy e uma Descida de Vizinhança Variável. A melhor variante de nosso método foi testada em um grande conjunto de instâncias e foi comparada favoravelmente com a melhor abordagem anterior no estado da arte.

Palavras-chave Gerenciamento de armazenagem · Problema de lote de pedidos online · Encomenda em lote · Tempo de giro · Heurísticas

1. Introdução

A separação de itens em um depósito, como parte da gestão da cadeia de abastecimento, segue uma política de separação que determina como a separação de itens é realizada no depósito. É possível encontrar diferentes políticas de separação na literatura, tais como: separação de pedido único,

Esta pesquisa foi parcialmente financiada pelos projetos: MTM2015-63710-P, RTI2018-094269-B-I00, TIN2015-65460-C2-2-P e PGC2018-095322-B-C22 do Ministerio de Ciencia, Innovación y Universidades (Espanha) ; pela Comunidade de Madrid e Fundo Europeu de Desenvolvimento Regional, Grant Ref. P2018 / TCS-4566; e pelo Programa Propio de I + D + i da Universidade Politécnica de Madrid (Programa 466A).

B Eduardo G. Pardo
eduardo.pardo@urjc.es

Informações estendidas sobre o autor disponíveis na última página do artigo

batching e sort-after-picking, picking de pedido único com zoneamento, batching com zoneamento, entre outros. O batch de pedidos pode ser considerado uma família de políticas de separação que se baseiam no agrupamento dos pedidos recebidos no depósito em lotes, antes de iniciar o processo de separação. Uma vez que os lotes tenham sido conformados, todos os itens nos pedidos do mesmo lote são separados na mesma rota de separação. Existem muitos problemas de otimização relacionados ao processo de separação de itens em um depósito quando a política de separação é o lote de pedidos.

Dentro desta família de problemas, a versão mais clássica é geralmente referida como Order Batching Problem (OBP) [44], que consiste em minimizar o tempo total necessário para coletar um grupo de pedidos recebidos em um depósito. Esta versão despertou um interesse relevante na comunidade científica. O OBP provou ser *NP*-difícil para casos gerais [12] No entanto, é solucionável em tempo polinomial se cada lote não contiver mais de dois pedidos [12] Infelizmente, as instâncias reais do warehouse geralmente não se enquadram nesta categoria. Conseqüentemente, ela tem sido abordada heuristicamente nos últimos anos por meio de heurísticas e metaheurísticas. A estratégia do First-Come First-Served (FCFS) pode ser a primeira abordagem heurística implementada em depósitos para atribuir pedidos a lotes. Essa estratégia tem sido amplamente utilizada devido à sua simplicidade. Outros métodos heurísticos importantes são *métodos de sementes* [13,22,32] e *métodos de salvamento* [40] Dentro [4] é possível encontrar um levantamento desses métodos onde os autores propuseram uma classificação.

Mais recentemente, também é possível encontrar abordagens baseadas em metaheurísticas na literatura. O primeiro algoritmo metaheurístico aplicado ao problema foi descrito em [23], onde um Algoritmo Genético é introduzido. Mais tarde, em [1], é proposto um algoritmo baseado na metodologia de Busca de Vizinhança Variável; especificamente, os autores consideraram vários bairros em um esquema Variable Neighbourhood Descent (VND). Henn et al. [19] propôs uma Pesquisa Local Iterada e um algoritmo de Sistema Ant baseado em classificação e em [21] Henn melhorou os resultados anteriores introduzindo dois algoritmos adicionais: Tabu Search e Attribute-Based Hill Climber. Dentro [30], Oncan propôs um algoritmo Iterated Local Search com um método Tabu Thresholding como procedimento de pesquisa local. Pelo que sabemos, a última proposta e o estado da arte atual para o problema foi feita em [26] onde os autores propuseram um algoritmo multi-start baseado na metodologia Variable Neighbourhood Search (VNS). Em cada iteração desse algoritmo, ele começa gerando uma solução inicial diferente, que é aprimorada usando um VNS básico. Em seguida, uma estratégia de pós-otimização baseada no VNS geral é aplicada.

O OBP foi estendido com a inclusão de diferentes restrições ou funções objetivo alternativas. Os dois mais pendentes são: o Problema de sequenciamento e lote de pedidos (OBSP), que introduz uma restrição relacionada à data de vencimento de cada pedido (ver por exemplo [20,25]). Além disso, o Problema Mín-Máx de Batching de Pedidos (OBP Mín-Máx) consiste em buscar um equilíbrio na carga de trabalho de um grupo de selecionadores, ao considerar vários selecionadores para coletar os lotes. Referimos o leitor a [11,28] para revisar o estado da arte desse problema.

Todas as abordagens anteriores, dentro da família Order Batching, podem ser consideradas como variantes estáticas. Porém, o desenvolvimento do e-commerce online e a redução dos prazos de entrega garantidos pelos vendedores fazem com que isso *estático* abordagem do OBP muito restritiva e irreal. Em contextos reais, os pedidos estão entrando continuamente no sistema e é necessário modificar a alocação do lote inicial, para poder atender os novos pedidos dentro da janela de tempo comprometida com o cliente. Essa versão dinâmica do problema é conhecida como Problema de lote de pedidos on-line (OOBP). Portanto, a lista de pedidos que chegam ao armazém é atualizada online e os algoritmos têm que criar lotes e roteiros sem ter a informação completa de todos os pedidos que precisam ser coletados em um dia útil, nem o horário de chegada

de pedidos futuros ainda não no armazém. Observe que uma diferença fundamental desse problema em comparação com sua versão estática é que o método de criação de lotes e rotas funciona o dia todo. Na versão estática, o sistema oferece uma solução inicial válida para toda a jornada de trabalho. Caso a lista de pedidos seja atualizada, o sistema deve resolver um novo problema desde o início com o conjunto de pedidos atual. Na abordagem dinâmica apresentada neste artigo, o sistema está funcionando continuamente e cada vez que uma nova ordem entra no sistema, ela é incorporada à lista de ordens e considerada imediatamente para a obtenção de soluções e fi cientes.

Como é o caso do OBP, na versão online também é possível encontrar diferentes variantes do problema. Até onde sabemos, a primeira contribuição relacionada ao OOBP pode ser rastreada em lote até 1997, quando uma variante do OOBP com múltiplos selecionadores foi abordada em [44] Neste artigo, a função objetivo consiste em minimizar o tempo de giro. Os autores propuseram um método FCFS simples como estratégia de batching e um algoritmo de roteamento Traversal (S-Shape) para determinar a rota para os selecionadores. Mais tarde, em 2007 [45] o OOBP foi estudado para armazéns de blocos múltiplos. Neste caso, a função objetivo abordada minimiza o tempo médio de processamento dos pedidos do cliente, que considera o tempo dedicado a três das principais atividades envolvidas no processo dos pedidos (batching, picking e triagem). Além disso, os autores deste artigo estudaram a influência de vários fatores do problema na determinação da solução ótima, como o tamanho do lote ou a alocação dos trabalhadores. Além disso, eles compararam diferentes estratégias para classificar os itens dentro do mesmo lote, como Classificar durante a coleta e Selecionar e classificar. Em 2011, Rubrico et al. [41] abordou o Problema de Reprogramação Online com vários selecionadores. Eles propuseram dois métodos heurísticos baseados na estratégia Steepest Descent Insertion e na estratégia MultistageRescheduling. Ambos combinados com o algoritmo de roteamento S-Shape. Nesse caso, os autores minimizaram a distância total máxima percorrida por cada selecionador. Uma combinação do estudo da influência de vários blocos e vários selecionadores é abordada em [2] Os autores estudaram a janela de tempo (janela de tempo fixo vs janela de tempo variável) que determina se é necessário esperar por novos pedidos ou tentar classificá-los novamente em lotes. Desta vez, os algoritmos de roteamento S-Shape e Largest Gap foram comparados, e a função objetivo é minimizar a distância total máxima percorrida por cada seletor. Outras variantes recentes do problema abordam oOOBP integrado com o agendamento da entrega, tentando minimizar o tempo de serviço de um pedido [47] Mais tarde, em [46] o mesmo problema foi resolvido, mas considerando vários selecionadores. Desta vez, também é estudada a carga de trabalho dos catadores. Dentro [48], novas restrições são introduzidas no mesmo problema. Desta vez, os autores consideraram várias zonas de entrega e um limite de capacidade no veículo usado para fazer a entrega.

A variante do OOBP estudada neste artigo foi abordada pela primeira vez em [18] Nesse caso, os autores consideraram um depósito de bloco único e apenas um selecionador. Para resolver o problema, eles propuseram um algoritmo heurístico baseado em Pesquisa Local Iterada (ILS) para minimizar o tempo máximo de conclusão dos pedidos do cliente, que chegam ao sistema em um determinado período de tempo. No entanto, outras funções objetivo, como o tempo máximo e o tempo médio que um pedido permanece no sistema, também foram relatadas. O método proposto foi comparado com um algoritmo FCFS clássico e com um método de Clarke e Wright. Esta variante do problema também foi abordada mais tarde em [34], onde os autores propuseram uma variante do conhecido algoritmo de estimativa de distribuição (EDA) para resolver o problema. Desta vez, os autores relataram a distância média percorrida pelo selecionador como função objetivo. No entanto, eles não compararam sua proposta com o algoritmo mais recente no estado da arte para o problema [18], mas com uma versão anterior dos mesmos autores baseada na metodologia Tabu Search proposta em [21] e originalmente projetado para a versão estática do problema. Adicionalmente,

o EDA proposto não foi capaz de melhorar os resultados da Pesquisa Tabu. Novamente, esses autores usaram o método S-Shape como estratégia de roteamento.

Está bem documentado que as operações de separação de pedidos são um dos processos mais importantes e caros em um depósito [3,7] Além disso, se as duas decisões relativas à separação do pedido (ou seja, lote e roteamento) forem consideradas simultaneamente, os benefícios associados podem ser substancialmente aumentados. De acordo com [6], é possível reduzir o tempo de viagem em até 35%, simplesmente projetando adequadamente as rotas dos selecionadores de pedidos. Portanto, um elemento-chave que afeta fortemente o desempenho desses algoritmos é a sequência usada por cada selecionador para recuperar os itens em cada lote. Este problema se classifica como Problema do caixeiro viajante Steiner (ver [5]) embutido em um espaço métrico de tipo especial com propriedades que podem ser exploradas para desenvolver heurísticas poderosas. Ratliff e Rosenthal [37] propôs um procedimento ótimo polinomial baseado em programação dinâmica para roteamento em um armazém retangular. O procedimento é computacionalmente eficiente para armazéns sem corredores cruzados; no entanto, a eficiência diminui quando o armazém tem corredores transversais e, em alguns casos, as rotas propostas parecem ser*ilógico* para os catadores que, como resultado, desviam suas rotas das especi fi cadas [12] Alternativamente, diferentes heurísticas de roteamento foram propostas na literatura (ver [15,35,39]). Petersen em [35] realizou uma série de experimentos numéricos para comparar seis métodos de roteamento, concluindo que *composto [*35,36] e *maior lacuna* foram as melhores opções entre os métodos estudados. Abordagens recentes [26] empiricamente descobriu que o *Combinado* método, originalmente proposto em [38] foi a estratégia com melhor desempenho em armazéns retangulares de bloco único.

Neste artigo, propomos um novo algoritmo para resolver o problema de envio em lote de pedidos online. Particularmente, nossa proposta é baseada em duas metaheurísticas bem conhecidas: Greedy Randomized Adaptive Search Procedure (GRASP) [10] e Descida de Vizinhança Variável [29] O primeiro é usado como uma estrutura geral para construir pontos de partida e fi cientes para o VND, que se encarrega de aprimorar a solução fornecida pelo GRASP. O VND usado aqui é uma implementação básica clássica da estrutura VNS. O método proposto supera as tentativas anteriores no estado da arte.

O resto do artigo está organizado da seguinte forma: na seção. 2 apresentamos a variante do problema OOBP abordada em detalhes. Na seção3 apresentamos os diferentes algoritmos propostos para resolver o problema. Seção4 apresenta os principais resultados computacionais obtidos, quando os algoritmos são aplicados a diferentes layouts de warehouse. Finalmente, as principais conclusões do trabalho e o esboço de planos de pesquisa futuros são coletados na Seção.5.

2 Definição de problema

Neste artigo, abordamos o problema de lote de pedidos on-line com um único selecionador em um depósito de um bloco. Como foi descrito na Seção1, este problema consiste em recolher todas as encomendas dos clientes que chegam a um armazém, minimizando uma função objetivo pré-definida.

Um pedido é uma lista de itens exigidos por um cliente que estão armazenados no depósito. Os pedidos são agrupados em lotes de uma capacidade máxima pré-definida, antes de serem coletados. Todos os pedidos de um mesmo lote são coletados juntos, pelo mesmo selecionador, em um único roteiro. Nesse sentido, um pedido não pode ser dividido em mais de um lote.

Uma questão importante, na variante abordada neste artigo, é que os pedidos que precisam ser gerenciados em um dia de trabalho, não estão totalmente disponíveis no início do dia, mas chegam ao depósito enquanto os catadores estão trabalhando. É por isso que o problema é considerado online.

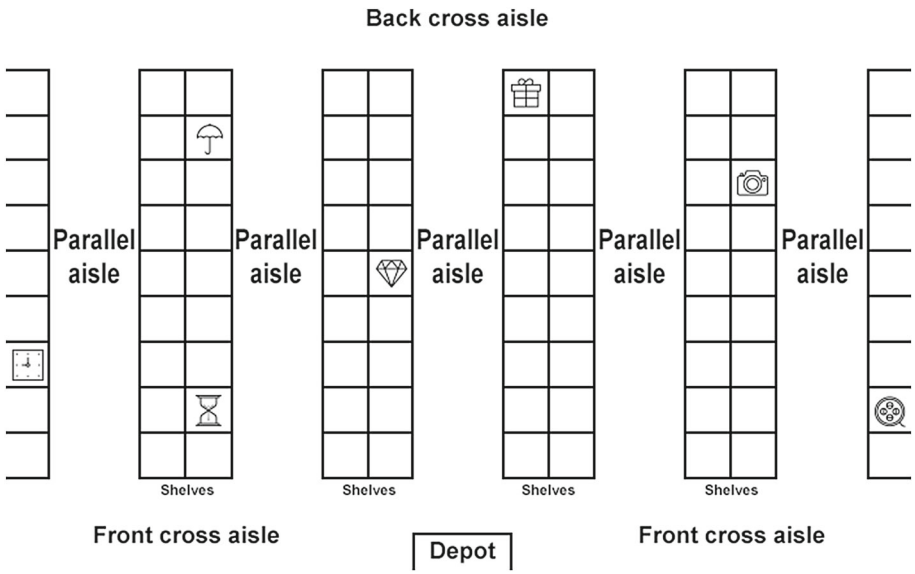


Figura 1 Layout de armazém

O OOBP aqui estudado considera apenas depósitos com layout retangular. Ou seja, o armazém é composto por dois corredores transversais (um na frente e outro atrás) e um número variável de corredores paralelos. Um exemplo desse layout de armazém é ilustrado na Fig.1. Neste exemplo, o armazém possui cinco corredores paralelos formados por prateleiras em cada lado do corredor, para armazenar itens. Particularmente, cada corredor paralelo consiste em 18 posições de coleta considerando as prateleiras em ambos os lados do corredor.

Os coletores iniciam e terminam suas rotas em um local específico do depósito, denominado depósito. Este depósito é o local onde os itens coletados devem ser entregues assim que forem recuperados. O depósito é sempre colocado no corredor transversal frontal, no meio do corredor ou no canto esquerdo.

O OOBP consiste em três tarefas diferentes: lote, seleção e roteamento. A tarefa de envio em lote consiste em agrupar os pedidos já disponíveis no sistema em lotes. Então, uma vez que os lotes estejam conformados, outro algoritmo deve selecionar qual, entre todos os lotes conformados, será coletado a seguir. Finalmente, uma rota para coletar os pedidos dentro do lote selecionado precisa ser construída. Um selecionador iniciará então o processo de recuperação de todos os itens do lote, seguindo o roteiro previamente construído. Quando o selecionador termina a coleta de itens e os entrega no depósito, um novo lote e rota são atribuídos ao selecionador para continuar seu trabalho.

Neste artigo consideramos a minimização de duas funções objetivo diferentes relacionadas ao OOBP: minimizar o tempo máximo de conclusão de todos os lotes e minimizar o tempo máximo de retorno de qualquer pedido. Essas funções objetivo foram relatadas anteriormente na literatura, no entanto, são consideradas separadamente (ou seja, não podem ser consideradas em um problema de otimização multiobjetivo, uma vez que não são necessárias em caso de conflito).

Para uma melhor compreensão dessas duas funções objetivo, na Fig. 2 nós mostramos a vida ciclo de um pedido o_{eu} através da linha do tempo. Observe que, como esse problema é considerado online, a linha do tempo não é limitada. Isso significa que os pedidos estão chegando ao sistema continuamente (ou seja, 24h por dia / 7 dias por semana) e apenas observamos o que acontece no sistema em um determinado

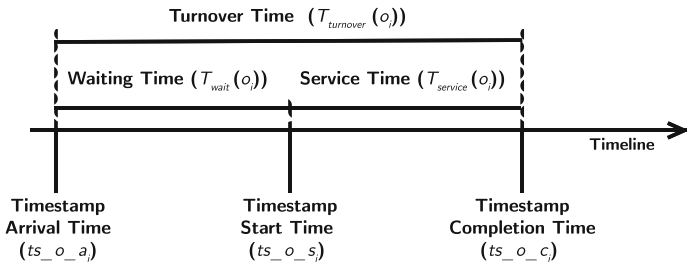


Figura 2 Ciclo de vida de um pedido *oi* através da linha do tempo

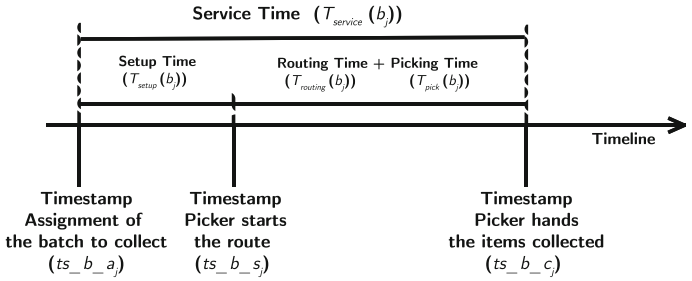


Fig. 3 Ciclo de vida de um lote *bj* através da linha do tempo

pedaço de tempo, para poder observar o comportamento de um algoritmo e compará-lo com outros algoritmos.

Nesta linha do tempo, destacamos três timestamps importantes (*ts)* no ciclo de vida de qualquer ordem o_{eu} : a hora de chegada ($ts\_o\_a_{eu}$), a hora de início ($ts\_o\_s_{eu}$), e o tempo de conclusão ($ts\_o\_c_{eu}$). A hora de chegada representa o momento em que a encomenda chega ao armazém. A hora de início representa o momento em que o selecionador começa a rota para coletar o lote que contém o pedido. Por fim, o tempo de conclusão representa o momento em que o lote, que contém a encomenda, é totalmente processado e os itens das encomendas recolhidas são entregues no armazém. Com esses três momentos no tempo disponível, também é possível definir três períodos de tempo no ciclo de vida da ordem: a espera Tempo, $T_{Cai\,t\,(Oeu)}$, o tempo de serviço, também conhecido como makespan, $T_{Servgelo(Oeu)}$, e o tempo de rotação, $T_{virarver\,(Oeu)}$. O tempo de espera representa o tempo que um pedido permanece no sistema antes de ser coletado. Pode ser calculado da seguinte forma: $T_{Cai\,t\,(Oi)} = ts\_o\_s_{eu} - ts\_o\_a_{eu}$. O tempo de serviço é o tempo necessário para coletar um pedido, uma vez que o selecionador tenha iniciado a rota. Isto pode ser calculado da seguinte forma: $T_{Servgelo(Oi)} = ts\_o\_c_{eu} - ts\_o\_s_{eu}$. Por fim, o tempo de giro é o tempo que um pedido permanece no sistema, seja aguardando ou sendo recebido. Pode ser calculado da seguinte forma: $T_{virarver\,(Oi)} = ts\_o\_c_{eu} - ts\_o\_a_{eu}$.

É importante destacar que, no contexto do OOBP, os pedidos não são coletados individualmente, mas eles são coletados em lotes (ou seja, um grupo de pedidos que são coletados juntos em uma única rota). Portanto, na Fig.3 representamos um esquema geral do ciclo de vida de um lote ao longo da linha do tempo.

Nesta linha do tempo, destacamos três timestamps importantes no ciclo de vida de qualquer lote b_j: a atribuição de um lote ao selecionador ($ts\_b\_a_j$), a hora de início quando o selecionador inicia a rota para coletar os itens nos pedidos atribuídos ao lote ($ts\_b\_s_j$), e a tempo de conclusão do lote ($ts\_b\_c_j$), quando o selecionador entrega os itens coletados na rota, para o depósito. Com esses três momentos no tempo disponível, também é possível

tabela 1 Parâmetros e variáveis para o OBP

Parâmetros		
n	→	Número de pedidos de clientes disponíveis no sistema em um determinado carimbo de data / hora
m	→	Limite superior do número de lotes (um valor simples é $m = n$)
$v_{roteamento}$	→	Velocidade de roteamento: número de unidades de comprimento que o selecionador pode percorrer no armazém por unidade de tempo
$v_{escolha}$	→	Número de itens que o selecionador pode pesquisar e escolher por unidade de tempo
C_{eu}	→	Número de itens do pedido o_i por $1 \le eu \le n$
C	→	Número máximo de artigos que podem ser incluídos em um
Variáveis		
x_{ji}	→	{lote (capacidade do dispositivo) 1, se pedido o_i é atribuído ao lote b_j, 0, caso contrário

definir diferentes períodos de tempo no ciclo de vida de qualquer lote: o tempo de serviço, $T_{Servgelo(b\ j)}$, o tempo de configuração ($T_{configurar)}$, o tempo de roteamento $T_{roteamento\ (b\ j)}$, e o tempo de colheita, $T_{escolha(b\ j)}$. O tempo de configuração representa o tempo que o sistema e o selecionador precisam para preparar a coleta carrinho, para receber e analisar a lista de pedidos atribuídos (o lote) e para realizar qualquer outra tarefa administrativa antes do início do percurso. Geralmente é considerado constante para qualquer lote e é definido na instância do problema. Uma vez que o selecionador está pronto para partir, os tempos de roteamento e coleta são os tempos necessários para a coleta percorrer os corredores, procurando os itens nos pedidos atribuídos e para extrair (pesquisar e selecionar) qualquer item das prateleiras, respectivamente. A soma dos tempos de configuração, roteamento e coleta é conhecida como tempo de serviço. Observe que o tempo de atendimento é o tempo disponível para que os algoritmos componham uma nova disposição de lotes, considerando apenas os pedidos que chegaram ao armazém em um momento anterior e que ainda não foram coletados. Portanto, o tempo de serviço pode ser calculado da seguinte forma:

$$T_{Servgelo(b\ j)} = T_{roteamento\ (b\ j)} + T_{escolha(b\ j)} + T_{configurar,}\ \forall\ j \in \{1,\ldots,m\}.$$

Em seguida, definimos formalmente o OBP com base na formulação apresentada em [18] Observe que o OOBP é igual ao OBP se considerarmos um determinado instante no tempo (ou seja, ele leva em consideração apenas os pedidos já no sistema, pressupondo que nenhum outro pedido chegará depois). Primeiro, na Tabela1, apresentamos os parâmetros e variáveis necessários para definir o problema corretamente. Em seguida, as funções objetivo e as restrições que definem as variantes do OBP abordadas neste artigo são apresentadas.

O tempo de roteamento é determinado pelo algoritmo de roteamento. Por uma questão de simplicidade, consideramos uma função d ($b\ j$) que recebe os pedidos no lote $b\ j$ e retorna a distância percorrida para coletar esses pedidos. Esta função depende da estratégia de roteamento que será apresentada em Sect. 3,3. Portanto, considerando uma velocidade $v_{roteamento,}$ o tempo de roteamento é definido da seguinte forma:

$$T_{roteamento\ (b\ j)} = \frac{d\ (b\ j)}{v_{roteamento}}, \quad \forall\ j \in \{1,\ldots,m\}.$$

Considerando que C_{eu} é o número de itens no pedido o_{eu} atribuído a $b\ j$, e $v_{escolha}$ é o número de itens que o selecionador pode pesquisar e escolher por unidade de tempo, o tempo de separação

para um lote b_j pode ser definido como segue:

$$T_{escolha(b_j)} = \sum_{i=1}^{n} \frac{ijj}{v_{escolha}}, \quad \forall\, j \in \{1,\ldots,m\}.$$

Finalmente, $T_{configurar,}$ o tempo de configuração é considerado um parâmetro e é especificado na instância particular.

O primeiro objetivo é minimizar o tempo máximo de conclusão de qualquer pedido e é dado por:

$$\min_{j \in \{..,m} \max\} \left(ts\_b\_s_j + T_{Servgelo(b_j)}. \right) \tag{1}$$

Vale ressaltar que esse objetivo é determinado pelo momento em que o colhedor entrega o último lote.

A segunda função objetivo considerada neste artigo é minimizar o máximo tempo de giro dos pedidos recebidos. O tempo de giro de um pedido o_{eu}, $T_{virarver\,(o_{eu}),}$ pode ser calculado da seguinte forma:

$$T_{virarver\,(oi)} = m \left(\sum_{j=1} ts\_b\_s_j + T_{Servgelo(b_j)} x_{ji} \right) - ts\_o\_a_{eu}, \forall\, eu \in \{1,\ldots,n\}.$$

E então, a segunda função objetivo pode ser expressa como:

$$\text{mínimo máximo } \} T_{virarver\,(o_{eu}).} \quad e \in \{, \ldots, n \tag{2}$$

Observe que o valor desta função objetivo é determinado pelo tempo de giro do pedido que permanece mais tempo no sistema.

O conjunto de soluções viáveis, em ambos os casos, é dado pelas seguintes restrições:

- Restrições em (3) garantia que cada pedido é atribuído a apenas um lote:

$$\sum_{j=1}^{m} x_{ji} = 1, \ \forall\, eu \in \{1,\ldots,n\}. \tag{3}$$

- Restrições em (4) garantem que a capacidade máxima de cada lote não seja excedida:

$$\sum_{i=1}^{n} C_{eu}\, x_{ji} \leq C, \ \forall\, j \in \{1,\ldots,m\}. \tag{4}$$

- Restrições em (5) garantir que o lote b_j começa a ser coletado, uma vez que o lote b_{j-1} foi coletado:

$$ts\_b\_s_j \geq ts\_b\_s_{j-1} + T_{Servgelo(b_{j-1}),} \ \forall\, j \in \{2,\ldots,m\}. \tag{5}$$

- Restrições em (6) garantir que a rota de coleta de um lote b_j não pode começar antes os carimbos de data / hora (momentos no tempo) quando os pedidos o_{eu} atribuído a esse lote ter chegou ao sistema:

$$ts\_b\_s_j \geq ts\_o\_a_{eu}\, x_{ji,} \ \forall\, eu \in \{1,\ldots,n\}, \text{ e } \forall\, j \in \{1,\ldots,m\}. \tag{6}$$

- Restrições em (7) e (8) afirmam a não negatividade de $ts\_b\_s_j$ e $ts\_o\_s_{eu}$, respectivamente:

$$ts\_b\_s_j \geq 0, \ \forall\, j \in \{1,\ldots,m\}. \tag{7}$$

$$ts\_o\_s_{eu} \geq 0, \ \forall\, eu \in \{1,\ldots,n\}. \tag{8}$$

Finalmente, as restrições em (9) afirmam que as variáveis x_{ji} são binários:

$$x_{ji} \in \{0, 1\}, \forall\, j \in \{1,\ldots,m\} \text{ e } \forall\, eu \in \{1,\ldots,n\}. \tag{9}$$

Apesar do fato de que este modelo, proposto em [18], representa o OBP em vez do OOBP, ajuda a entender o objetivo do OOBP. Além disso, esta formulação é um modelo de programação inteira mista não linear que não pode ser usado para resolver instâncias reais usando um solver. Isso suporta parcialmente a adequação do uso de algoritmos heurísticos neste contexto, como propomos neste artigo.

É importante ressaltar que, no caso do turnover time, a comparação do valor da função objetivo de duas soluções apresenta algumas dificuldades adicionais. Vamos apresentar o conceito de intervalo de tempo como o tempo entre duas partidas consecutivas do selecionador. Note que a construção de uma solução no contexto do OOBP em um determinado momento do tempo consiste em: i) agrupar os pedidos disponíveis no armazém em lotes, e ii) determinar a sequência em que esses lotes devem ser recolhidos. Em seguida, essa solução parcial é avaliada e comparada com outras soluções parciais comparáveis (ou seja, aquelas calculadas no mesmo intervalo de tempo). Quando o selecionador está pronto para uma nova partida, o primeiro lote classificado na sequência da melhor solução encontrada durante o intervalo de tempo é atribuído ao selecionador.

Qualquer pedido recém-chegado ao depósito deve esperar até que a solução parcial em construção seja concluída. Então, ele pode ser incluído no processo de construção da próxima solução parcial (não importa se o intervalo de tempo não terminou). Além disso, uma vez que todos os pedidos tenham sido atribuídos a um lote, o lote contendo o pedido não coletado mais antigo (ou seja, o primeiro pedido chegou ao armazém entre os não coletados) é selecionado para ser atribuído ao selecionador uma vez no próximo slot de tempo começa.

3 algoritmos

Nesta seção, descrevemos nossa proposta algorítmica para lidar com o OOBP. Primeiro, na seção3,1 descrevemos o método responsável pela simulação dos processos gerais que acontecem no armazém. Esses métodos incluem: considerar os pedidos fornecidos por um despachante; realizar o batching dos pedidos considerados; escolher o próximo lote a ser coletado; determinar a rota para coletar o lote; e determinar o momento de partida do selecionador. Assim que o esquema geral estiver disponível, na seção3,2 apresentamos os algoritmos propostos para lidar com a tarefa de batching. Esta seção está dividida em Seita.3.2.1 onde apresentamos o Procedimento de pesquisa adaptativa aleatória gananciosa [10] como o método construtivo, e Sect. 3.2.2 onde apresentamos a Descida de Vizinhança Variável [29] usado como uma pesquisa local no GRASP. Finalmente, na seção3,3, detalhamos a estratégia de roteamento usada.

3.1 Esquema geral

As funções objetivo da OOBP estudadas neste trabalho consistem em minimizar o tempo total decorrido na coleta de todos os pedidos que chegam a um armazém em um horizonte de tempo pré-definido ou, alternativamente, minimizar o tempo de giro. Devido à natureza online deste problema, para calcular essas funções objetivo, é necessário ter um algoritmo geral capaz de gerenciar todos os processos envolvidos. Chamamos esse algoritmo de método de orquestração e é apresentado em Algoritmo1.

O algoritmo de orquestração recebe dois parâmetros de entrada: o horizonte de tempo para a recepção de pedidos (*maxT ime)* e a lista de pedidos pendentes no início do processo (*lista de pedidos).* Observe que essa lista de pedidos, em um sistema online, pode se referir a pedidos recebidos no período noturno ou pendentes de cobrança do dia útil anterior. No entanto, consideramos que todo o trabalho do dia anterior já foi feito.

O método é executado enquanto há pedidos pendentes para serem coletados ou o tempo máximo permitido não foi atingido (etapa 4) Em seguida, ele verifica se há novos pedidos que chegaram ao sistema (etapa5) e atualiza a lista de pedidos pendentes. Assim que esta lista for atualizada, o algoritmo de envio em lote é executado (etapa6) e a nova solução é comparada com a melhor solução encontrada até o momento. Então, se houver um selecionador disponível e a solução contiver lotes ainda não coletados (etapa8), o lote mais adequado da melhor solução (*melhor solução)* é escolhido e o algoritmo de roteamento (passo 10) construirá uma rota para coletar o lote. Em seguida, o selecionador irá coletar todos os pedidos dentro do lote selecionado (etapa11) Por fim, a lista de pedidos pendentes é atualizada, retirando-se os pedidos coletados (passo12) Observe que o algoritmo não espera até que o selecionador volte de sua rota, mas está continuamente em execução para ter a melhor solução possível disponível assim que o selecionador estiver disponível novamente.

Algoritmo 1 Método de orquestração

```
1: Procedimento Orquestração(maxT ime, listar pedidos)
2: ordens pendentes ← listar pedidos
3: melhor solução ← ∅
4: enquanto (getTime () < maxT ime) || (pendentesOrdens =∅) Faz
5:     ordens pendentes ← ordens pendentes ∪ getNewOrders ()
6:     solução ← batchingAlgorithm (ordens pendentes)
7:     atualizar(melhor solução, solução)
8:     E se isPickerAvailable () então
9:         lote ← selectBatchAlgorithm (melhor rota de solução) ←
10:        routingAlgorithm (lote)
11:        recolher (lote, rota)
12:        remover(pendentesOrdens, lote)
13:    fim se
14: terminar enquanto
```

Existem três métodos notáveis dentro do procedimento de orquestração apresentado no Algoritmo 1. O algoritmo de envio em lote (batchingAlgorithm), que conforma os lotes a serem coletados, é descrito na seção. 3,2. O algoritmo de roteamento (routingAlgorithm), que determina a rota que um selecionador deve seguir para coletar um lote, é descrito na Seção 3,3. Finalmente, o algoritmo de seleção (selectBatchAlgorithm), que determina o próximo lote da solução a ser atribuído a um selecionador disponível. Nesse caso, não dedicamos uma seção inteira ao algoritmo, pois ele segue uma heurística muito simples. Particularmente, este método seleciona o lote que contém o pedido mais antigo no sistema. Este método é comumente denominado como FIRST na literatura relacionada [18]

3.2 Algoritmo de lote

Como foi mencionado anteriormente, o batching é um dos procedimentos-chave no contexto do OOBP. Consiste em agrupar todos os pedidos recebidos em um depósito em um conjunto de lotes de tamanho máximo pré-definido com o objetivo de minimizar uma determinada função objetivo (tipicamente a distância necessária para coletar todos os pedidos).

Neste artigo, propomos um algoritmo de batching baseado na combinação de um procedimento GRASP (apresentado na Seção 3.2.1 e usado como um método construtivo) e um VND (apresentado na Seção 3.2.2 e usado como uma pesquisa local dentro do GRASP).

3.2.1 Procedimento de pesquisa adaptativa aleatória gananciosa

O Greedy Randomized Adaptive Search Procedure foi introduzido em [10] como um método multistart para encontrar soluções de alta qualidade para problemas de otimização complexos. Cada iteração é composta de duas etapas: (1) construção e (2) melhoria. O esquema geral do GRASP é apresentado no Algoritmo2. A fase de construção combina a ganância de uma função gananciosa particular e a aleatorização de algumas decisões durante a construção. O procedimento construtivo no GRASP (etapa4) proposto neste artigo é explicado a seguir. Por outro lado, a fase de melhoria (etapa5), geralmente baseado em um método de pesquisa local, neste caso é baseado em um procedimento metaheurístico. Particularmente, substituímos a pesquisa local por um método VND, explicado em detalhes na Seção3.2.2. Portanto, em cada iteração, o método GRASP constrói uma solução eficiente, e essa solução é ainda melhorada com um procedimento VND. Este esquema GRASP é repetido até que o método esgote o tempo ou o número máximo de iterações seja alcançado.

Algoritmo 2 Procedimento de pesquisa adaptativa aleatória gananciosa

1: **Procedimento** APERTO(*maxT ime)*
2: *melhor solução* ← ∅
3: **repetir**
4: *solução* ← Construtivo()
5: *solução*′ ← LocalSearch (*solução*)
6: *melhor solução* ← Atualizar(*melhor solução, solução*′)
7: **até** getTime ()> *maxT ime*
8: **Retorna** *melhor solução*

O método construtivo GRASP proposto neste artigo é apresentado em Algoritmo.3 e é baseado em dois princípios básicos: uma função gananciosa que seleciona um grupo de itens candidatos a serem adicionados à solução na próxima iteração e uma aleatorização particular das decisões tomadas por essa função. O método começa a partir de uma solução vazia (etapa2) e, em cada iteração, adiciona um novo pedido à solução. Todos os pedidos disponíveis são inicialmente inseridos na chamada Lista de Candidatos (*CL) (*Passo 3) Propomos o uso de uma função gulosa (f) com base no peso dos pedidos no *CL* de forma que a ordem mais pesada seja considerada em primeiro lugar (os empates são quebrados aleatoriamente). Portanto, o GRASP construtivo ordena todos os pedidos, já no sistema, mas ainda não atribuídos a um lote, de forma decrescente em relação ao seu peso. Então, um limite ^{o} é calculado (passo 6) com base no máximo (argmax f (*CL))* e mínimo (argmin f (*CL))* peso de qualquer pedido no *CL,* e um valor aleatório $\alpha \in$ *VOCÊ*[0, 1] (passo 4) Este limite é usado para determinar a porcentagem dos melhores candidatos a serem incluídos em uma nova lista, chamada Lista Restrita de Candidatos (*RCL) (*Passo 7) Finalmente, um pedido é escolhido aleatoriamente a partir deste*RCL (*Passo 8), e é incluído na solução parcial que está sendo construída nesta iteração. O pedido escolhido é inserido no primeiro lote com espaço disponível suficiente (etapa9) e é removido do *CL (*Passo 10)

Observe que, uma vez que um novo pedido é adicionado à solução, é necessário determinar o lote onde será alocado. Na primeira iteração, um novo lote vazio é criado. Então, nas seguintes iterações, o algoritmo tenta inserir a ordem selecionada neste lote e

se não couber no lote, outro lote vazio é criado para alocar esse pedido e assim por diante. Observe que a sequência de lotes usada para tentar a inserção do pedido é a mesma sequência em que os lotes são criados.

Os critérios de parada clássicos para o GRASP estão relacionados a um determinado número de iterações ou a um horizonte de tempo pré-definido. No entanto, neste artigo, executamos o método GRASP, desde que não haja um selecionador ocioso. Isso significa que cada vez que um novo lote é atribuído a um selecionador, o procedimento GRASP é executado novamente, mas desta vez não considera nenhum pedido que está sendo coletado (ou seja, está no lote atribuído) ou foi coletado antes.

Vale ressaltar que devido ao contexto online deste problema, o *CL* está sendo atualizado toda vez que o procedimento GRASP é reiniciado, com os últimos pedidos que chegam ao sistema.

Algoritmo 3 Procedimento construtivo

```
1: Procedimento Construtivo(listar pedidos)
2: solução ← ∅
3: CL ← listar pedidos
4: α ← getRandomValue ()
5: enquanto CL = ∅ Faz
6:       º← argmax f (CL) - α (argmax f (CL) - argmin f (CL))
7:       RCL ← buildRestrictedCandidateList (th, CL) pedido ←
8:       randomOrderSelection (RLC)
9:      insertOrder (solução, pedido) CL ←
10:      CL \ {pedido}
11: terminar enquanto
12: Retorna solução
```

3.2.2 Descida de Vizinhança Variável

Variable Neighborhood Search é uma metaheurística proposta por Mladenović e Hansen em 1997 como um método geral para resolver problemas de otimização complexos. Os autores introduziram a ideia de mudar a estrutura da vizinhança dentro da pesquisa para alcançar diferentes ótimos locais. Existem muitas variantes do VNS. Alguns dos mais conhecidos são: VNS reduzido (RVNS), Descida de vizinhança variável (VND), VNS básico (BVNS), VNS geral (GVNS), SkewedVNS (SVNS) e Pesquisa de decomposição de vizinhança variável (VNDS) [16,17,29] No entanto, abordagens mais recentes surgiram nos últimos anos, tais como: Variable Formulation Search (VFS) [33], Pesquisa de vizinhança de variável paralela (PVNS) [9,28], ou Pesquisa de vizinhança de variável multi-objetivo (MO-VNS) [8]

Neste artigo, propomos o uso de um procedimento VND como uma busca local dentro da metodologia GRASP. O VND foi proposto no contexto da Pesquisa de Vizinhança Variável em [29], como estratégia geral para explorar sistematicamente um grupo de bairros. O resultado obtido de um procedimento VND é um ótimo local com respeito a todas as estruturas de vizinhança consideradas. A ordem final dos bairros estudados determina o desempenho do método. Normalmente, as estruturas de vizinhança são classificadas da menor e mais rápida para explorar, até a maior e mais lenta para explorar. No entanto, esta regra deve ser testada empiricamente ao considerar um problema particular e as estruturas de vizinhança associadas.

Usamos uma implementação padrão e básica de VND, que pode ser considerada como o método VND clássico. Em Algoritmo4 apresentamos o pseudocódigo do procedimento VND proposto neste artigo. Particularmente, este algoritmo recebe uma solução inicial como ponto de partida

e considera três estruturas de vizinhança diferentes (nomeadas no pseudocódigo como N_1, N_2, e N_3) explorado por um procedimento de pesquisa local. Este procedimento de busca local segue uma primeira estratégia de melhoria. Os bairros específicos propostos são detalhados a seguir. O método irá inicialmente explorar a primeira vizinhança (N_1) no passo 7 e então irá determinar se uma melhoria foi feita (etapa 13) ou não. Se um bairro não consegue melhorar a solução atual, então o método irá pular para a próxima vizinhança disponível (passo 17) até que o número máximo de bairros seja alcançado (etapa 19) Quando a exploração de uma vizinhança melhora a solução atual, a melhor solução encontrada é atualizada e o método começa novamente a partir da primeira vizinhança (passo15)

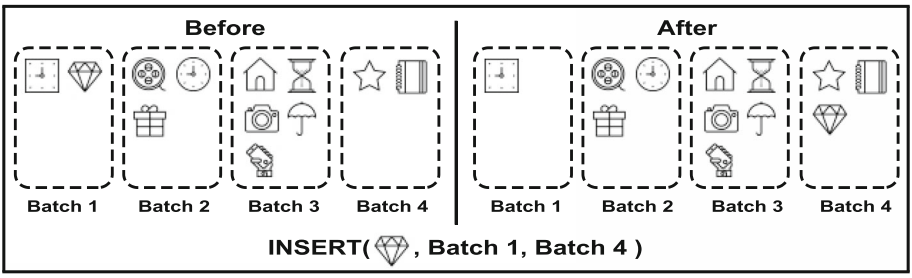
Algoritmo 4 Descida de Vizinhança Variável

```
1: Procedimento VND (solução)
2: k ← 1
3: kmax ← 3
4: melhor solução ← solução
5: repetir
6:     E se k == 1 então
7:         solução' ← LocalSearch (melhor solução, N1)
8:     mais se k == 2 então
9:         solução' ← LocalSearch (melhor solução, N2)
10:    mais se k == 3 então
11:        solução' ← LocalSearch (melhor solução, N3)
12:    fim se
13:    E se Avalie(solução') < Avalie(melhor solução) então
14:        melhor solução ← solução'
15:        k = 1
16:    senão
17:        k = k + 1
18:    fim se
19: até k> kmax
20: Retorna melhor solução
```

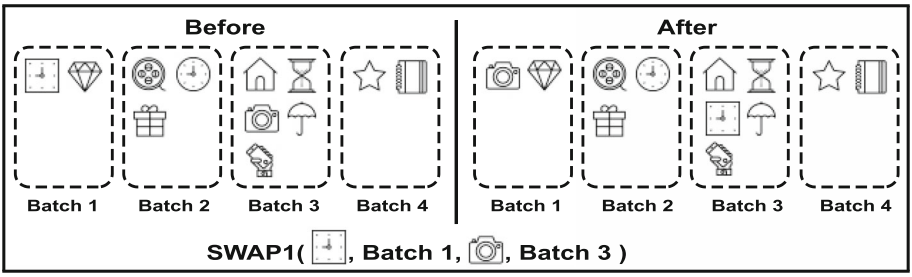
Propomos neste artigo três estruturas de vizinhança diferentes para lidar com o OOBP. Essas estruturas de vizinhança são nomeadasInserir, Trocar1, e Swap2 respectivamente, e eles são mostrados graficamente na Fig. 4. OInserir vizinhança, representada por um exemplo na Fig. 4a, considera tudo possível soluções alcançadas pela inserção de qualquer pedido na solução em todos os lotes disponíveis. No exemplo representado na Fig.4a é representado pela inserção de um diamante (originalmente alocado no Lote 1) no Lote 4. Descrevemos a configuração dos lotes antes e depois do Inserir Operação.

O Swap1 vizinhança, representada por um exemplo na Fig. 4b, considera tudo possível soluções alcançadas pela troca de qualquer par de pedidos em um lote diferente na solução, em todos os lotes disponíveis. No exemplo representado na Fig.4b é representada a troca de um relógio (originalmente alocado no Lote 1) por uma câmera fotográfica (originalmente alocada no Lote 4). Descrevemos a configuração dos lotes antes e depois doSwap1 Operação. finalmente, o

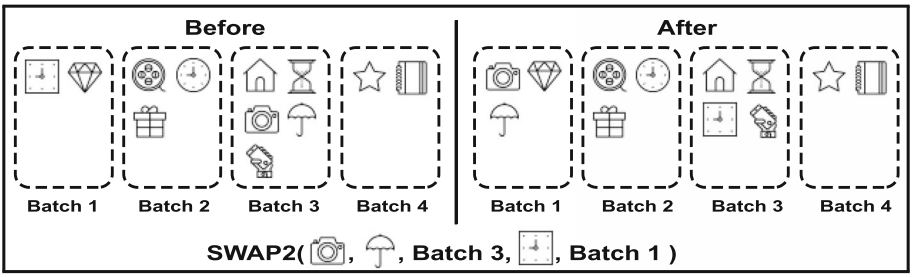
Swap2 vizinhança, representada por um exemplo na Fig. 4c, considera tudo possíveis soluções alcançadas pela troca de cada par de dois pedidos dentro do mesmo lote, com qualquer pedido único em qualquer outro lote. No exemplo representado na Fig.4c é representada a troca de uma câmera fotográfica e um guarda-chuva (originalmente alocado no Batch 3) com um relógio (originalmente alocado no Lote 1). Descrevemos a configuração dos lotes antes e depois doSwap2 Operação.



uma Insira 1x0.



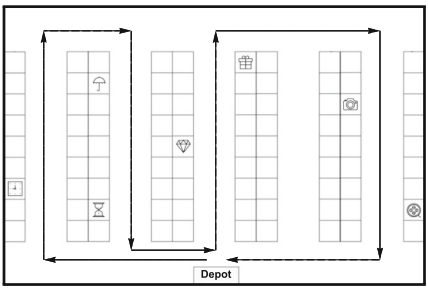
b Swap1 1x1.



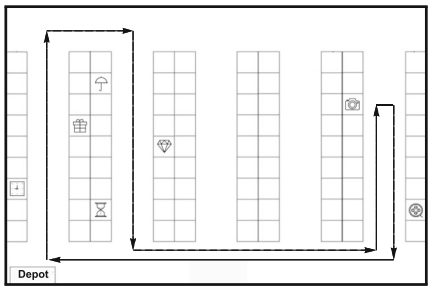
c Swap2 2x1.

Fig. 4 Estruturas de vizinhança

Observe que é obrigatório que os lotes resultantes não violem a restrição de capacidade máxima em nenhum lote. Caso contrário, a operação é considerada inviável e, neste caso, a solução obtida após uma mudança não é considerada como parte do bairro estudado.



uma Layout do armazém com o depósito colocado no centro do corredor transversal frontal.



b Layout do armazém com o depósito colocado no canto esquerdo do corredor transversal frontal.

Fig. 5 Exemplos de rota calculados com a estratégia S-Shape

3.3 Algoritmo de roteamento

O algoritmo de roteamento determina a rota que um selecionador deve seguir para coletar todos os itens dentro dos pedidos do mesmo lote. Esta rota sempre começa e termina no mesmo ponto, o depósito. O depósito é colocado no corredor transversal frontal, no centro ou no canto esquerdo do corredor.

Na literatura, há uma variedade de algoritmos para resolver o problema de roteamento em um warehouse. Este problema pode ser considerado uma variante do conhecido Problema do Caixeiro Viajante. Existem algoritmos heurísticos, metaheurísticos e exatos para resolver o problema. Em nosso caso, usamos o método heurístico S-Shape introduzido em [5,14] que é amplamente utilizado na literatura do OBP e OOBP devido à sua implementação simples e rápido desempenho. Além disso, as rotas obtidas são facilmente entendidas pelos catadores.

Dado um lote, o método S-Shape identifica os corredores paralelos onde há itens para coletar (pelo menos um item). Em seguida, cada um desses corredores é completamente atravessado de um corredor transversal (seja o corredor transversal frontal ou posterior) para o corredor transversal oposto. O primeiro corredor paralelo a ser percorrido é o corredor mais à esquerda que contém um item a ser coletado. Em seguida, o selecionador entrará apenas nos corredores paralelos que contenham pelo menos um item que precise ser coletado. Este processo é repetido até que o último corredor com itens seja atravessado. Em seguida, o selecionador retorna ao depósito usando o corredor transversal frontal. Observe que se o selecionador tiver que percorrer um número ímpar de corredores paralelos, o selecionador entrará no último corredor a partir do corredor transversal frontal. Nesse caso,

Na Fig. 5mostramos um exemplo de duas rotas diferentes projetadas usando o algoritmo S-Shape por meio de um armazém retangular. Particularmente, inFig.5a é mostrado um exemplo de layout da Câmara de Conhecimentos com o depósito colocado no centro do corredor transversal frontal. Nesse caso, a rota determinada pelo algoritmo S-Shape atravessa 4 corredores diferentes antes de retornar ao depósito. Na Fig.5b é mostrado um exemplo diferente de um layout de depósito onde o depósito é colocado no canto esquerdo do corredor transversal frontal. Nesse caso, o selecionador deve percorrer um número ímpar de corredores paralelos. Em seguida, os dois primeiros corredores paralelos são totalmente percorridos e, no último, o selecionador dá meia-volta ao chegar ao último item a coletar.

4 resultados computacionais

Nesta seção apresentamos os resultados computacionais obtidos com os algoritmos propostos na Seção. 3. Primeiro, na seção4,1 apresentamos as instâncias utilizadas na experimentação. Em seguida, descrevemos na seção4,2 a con fi guração do despachante de pedidos que determina o momento no tempo de chegada de cada pedido ao sistema. Na seção4,3 realizamos um conjunto de experimentos preliminares para ajustar os parâmetros dos métodos propostos e, adicionalmente, mostrar o mérito das diferentes partes do algoritmo final. Finalmente, na seção 4,4 comparamos nossa melhor variante com o estado da arte atual do OOBP para a versão abordada do problema.

Os experimentos foram executados em uma máquina Intel (R) Core (TM) 2 Quad CPU Q6600 2.4 Ghz, com memória RAM de 4 GBDDR2. O sistema operacional utilizado foi o Ubuntu 18.04.1 LTS de 64 bits, e todos os códigos foram desenvolvidos em Java 8.

4.1 Instâncias

Para testar os algoritmos propostos neste artigo, selecionamos dois conjuntos de instâncias derivadas da literatura anterior no estado da arte doOOBP. Observe que uma instância para oOOBP é formada por um grupo de parâmetros derivados de: as características do armazém, a lista de pedidos e o programador de chegada. O armazém também é definido por um determinado grupo de parâmetros como: o número de corredores, o comprimento de cada corredor, a política de triagem, etc. A lista de pedidos indica os produtos que devem ser coletados para atender a demanda dos clientes. Por fim, o programador de chegada determina o momento em que um determinado pedido chega ao depósito.

Todas as instâncias dentro dos conjuntos de dados selecionados apresentam características comuns, no que diz respeito ao formato do armazém: forma retangular, um bloco com dois corredores transversais (um na frente, um atrás) e um número variável de corredores paralelos. Esta distribuição de armazém é a mesma apresentada anteriormente na Fig.1. No entanto, apesar de todas as instâncias terem a mesma estrutura de armazém, consideramos diferentes tipos de armazém, que diferem em outros parâmetros, tais como: o número de corredores paralelos, o comprimento dos corredores, a largura dos corredores, o número de separação posições por corredor, etc. Para cada tipo de armazém, adicionalmente, estudamos diferentes listas de pedidos. Essas listas variam no número de pedidos e cada pedido varia no número e na composição dos itens.

O primeiro conjunto de dados usado na experimentação foi introduzido em [1] e foi originalmente definido para a versão estática do OBP. O resumo das características deste conjunto de dados é relatado na Tabela2. Este conjunto de dados inclui quatro configurações de warehouse diferentes (denotadas como W1, W2, W3 e W4). Para cada configuração de armazém, existem listas de pedidos cujo tamanho varia de 50 a 250 pedidos. Do número total de instâncias originalmente propostas pelos autores (2.400), uma seleção de instâncias foi feita em [26] para estabelecer um conjunto de dados reduzido. Neste artigo, os autores descobriram que o uso de todo o conjunto de dados não forneceu diferenças significativas quando comparado ao uso de apenas uma seleção reduzida de instâncias. Neste artigo, usamos 64 instâncias do conjunto de dados reduzido para realizar nossos experimentos.

O segundo conjunto de dados usado em nossos experimentos foi originalmente proposto em [18] Este conjunto de dados é composto por 1600 instâncias. Novamente, uma seleção de instâncias foi feita em [26] para ter um conjunto de dados reduzido que possa ser tratado em um período de tempo razoável. Nesse caso, também selecionamos 64 instâncias diversas para nossos experimentos. Na tabela3 apresentamos as principais características associadas a este armazém (W5). Desta vez, o tamanho das listas de pedidos varia de 40 a 100 pedidos.

mesa 2 Características do armazém [1]

	W1	W2	W3	W4
Política de armazenamento	Aleatório / ABC			
Posição do depósito	Centro / esquina			
Tamanho do pedido	U (1, 7)	U (2, 10)	U (5, 25)	U (1, 36)
Peso do Item	1	1	1	U (1, 3)
Capacidade do seletor de pedidos	12	24	150	80
(peso) Número de corredores	4	10	25	12
paralelos Número de itens por	2 × 30	2 × 20	2 × 25	2 × 16
corredor Número total de itens	240	400	1250	384
Comprimento do corredor paralelo (m)	50	10	50	80
Distância central entre dois corredores (m)	4,3	2,4	5	15

Tabela 3 Armazém características [18]

	W5
Política de armazenamento	Aleatório / ABC
Posição do depósito	Centro
Tamanho do pedido	U (5, 25)
Peso do Item	1
Capacidade do seletor de pedidos	30/45/60/75
(peso) Número de corredores	10
paralelos Número de itens por	2 × 45
corredor Número total de itens	900
Comprimento do corredor paralelo (m)	45
Distância central entre dois corredores (m)	5

Observe que os dois conjuntos de dados selecionados foram amplamente usados no contexto do OBP [4,24-28,42,43] e o OOBP [34,47,49] Além disso, é importante notar que os experimentos no contexto do OOBP são realizados em tempo real. Isso significa que o horizonte de tempo da chegada dos pedidos é o tempo mínimo que o algoritmo precisa para ser executado. Portanto, no contexto da OOBP não é possível considerar um número muito grande de instâncias nem horizontes de tempo muito longos para a chegada de pedidos. Na tabela4 descrevemos todos os parâmetros importantes usados em nossa experimentação. Entre outros, incluímos os parâmetros relacionados com a chegada das encomendas ao armazém. Esta configuração é comum para os 2 conjuntos de dados considerados. Em suma, o horizonte de tempo para a chegada das ordens foi definido em 4h e a distribuição estatística dos instantes de tempo das chegadas é determinada por uma distribuição exponencial.

Ao considerar o tempo de giro (tempo máximo que uma ordem permanece no sistema) como função objetivo, é necessário definir um momento no tempo como tempo de referência. Isso permite que o algoritmo calcule a quantidade de tempo de espera de uma ordem particular no sistema em relação a esse momento no tempo. Caso contrário, como o algoritmo está continuamente explorando novas soluções, não é possível comparar uma solução com outra obtida em um momento anterior, pois o ponto de referência no tempo varia. Esse tempo de referência é fixado e atualizado cada vez que um selecionador recebe um novo lote para coletar.

Tabela 4 Configuração dos parâmetros no experimentos

Período de chegada	4h
Distribuição do despachante Velocidade	Exponencial
do selecionador (LU / min) Tempo de	48
seleção por item (itens / min) Tempo de	6
configuração (min)	3
Janela de tempo ajuste (s)	400
Ordem inicial na fila	0
Semente aleatória	50
Estratégia de roteamento	Forma de S
Comece escolhendo a estratégia	Assim que o seletor estiver disponível
Estratégia de seleção de lote	Lote com o mais antigo pedido

Tabela 5 Valores lambda

# Pedidos	40	60	80	100	150	200	250
λ	0,167	0,250	0,333	0,417	0,625	0,833	1.042

4.2 Expedidor de pedidos

A chegada de pedidos ao armazém no contexto do OOBP é distribuída por um determinado horizonte de tempo. Nesse sentido, é necessário configurar um ambiente de simulação, que disponibilize os pedidos ao sistema antes de calcular a configuração do lote e posteriormente a rota de picking. O horizonte temporal de chegada das encomendas ao armazém está definido em 4h.

Para simular o tempo de chegada de cada pedido, seguimos um processo de ponto de Poisson. Uma vez que o horizonte de tempo está definido para 4h ($t = 4$). O número de eventos no intervalo de comprimento t é uma variável aleatória de Poisson $X(t)$ com média $E[X(t)] = \lambda * t$. O λ valor é selecionado dependendo do número de pedidos considerados no experimento. Neste caso, oλ valores escolhidos para nossos experimentos são compilados na Tabela 5.

4.3 Experimentos preliminares

Para determinar a melhor configuração do nosso algoritmo e, também, para testar a contribuição de cada parte do algoritmo, realizamos um conjunto de experimentos preliminares. Para isso, selecionamos um subgrupo de instâncias do total de conjuntos de dados. Particularmente, selecionamos 16 de 128 instâncias para realizar os experimentos preliminares. Estas instâncias foram selecionadas de forma a incluir no conjunto de dados reduzido as mais diversas instâncias.

4.3.1 Determinação dos melhores parâmetros GRASP

O algoritmo construtivo GRASP introduzido na Seção 3 tem dois parâmetros principais: (i) o valor do α parâmetro, e (ii) o número de construções. Quanto ao valor deα está preocupado, ele determina a voracidade na seleção de pedidos a serem incluídos na Lista de Candidatos Restritos. Neste artigo, usamos um critério ganancioso com base no peso dos pedidos (o pedido mais pesado é considerado primeiro).

Tabela 6 Desempenho de GRASP com diferentes α valores

	0,0	0,2	0,4	0,6	0,8	1.0	Aleatório
Dev. (%)	0,84	0,64	0,59	0,53	0,62	1.03	0,42
# Melhor	3	3	1	3	1	1	4

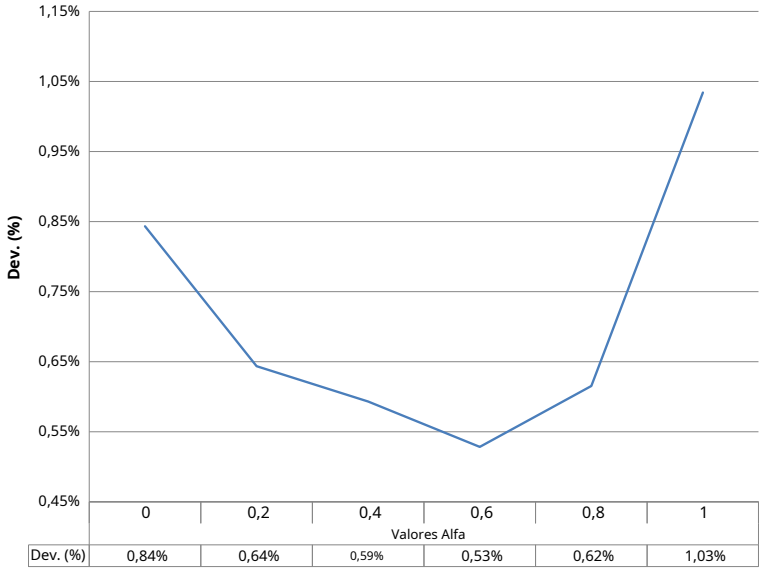


Fig. 6 Representação do desvio médio obtido com diferentes α valores

No primeiro experimento preliminar, testamos diferentes valores de α em ordem valor mais adequado. Na tabela6 relatamos o desvio médio em relação encontrado no experimento, e o número de melhores soluções encontradas para diferentes valores do parâmetro α. Observe que quando $\alpha = 1.0$ a inclusão de pedidos na Lista de Candidatos Restritos é totalmente aleatória. Por outro lado, quando$\alpha = 0.0$, a inclusão de pedidos na Lista de Candidatos Restritos é totalmente gananciosa. O desvio obtido com os diferentes valores deα é (Alfa) representado na Fig. 6. Como se mostra na figura, a evolução do desvio médio para o melhor valor encontrado diminui até$\alpha = 0,6$ e aumenta a partir deste ponto. No entanto, as diferenças entre os diferentes valores deα estudados são muito pequenos. Portanto, também incluímos na Tabela6 os resultados obtidos quando α assume um valor diferente e aleatório em cada iteração GRASP. Como é possível observar na tabela, as melhores soluções encontradas no experimento são obtidas quandoα é definido com um valor aleatório. Observe que este experimento foi realizado executando o algoritmo construtivo GRASP isoladamente por um limite de tempo de 60s. Para determinar o para o melhor valor

A seguir, testamos a influência do segundo parâmetro relacionado ao GRASP (isto é, número de construções). Neste caso, não nos interessa fixar o número de construções do algoritmo, pois dependerá do tempo disponível entre as rotas do selecionador. No entanto, é interessante revisar a influência do procedimento construtivo na qualidade da solução inicial.

Na tabela 7 relatamos a evolução do desvio do método construtivo GRASP em relação ao melhor valor do experimento, ao considerar de 10 a 10.000.000 de

Tabela 7 Evolução do desvio ao aumentar o número de construções GRASP

	10	100	1.000	10.000	100.000	1.000.000	10.000.000
Dev. (%)	4,66	3,22	2,60	1,82	1.07	0,56	0,00
# Melhor	0	0	0	0	0	1	15
CPUt (s)	0,01	0,02	0,12	0,92	9,11	90,35	900,96

Tabela 8 Ordens de vizinhança diferentes dentro do VND

	$\{N_3, N_1, N_2\}$	$\{N_3, N_2, N_1\}$	$\{N_3, N_2, N_1\}$	$\{N_2, N_1, N_3\}$
Dev. (%)	0,67	1,11	0,93	1.06
# Melhor	4	3	4	6
CPUt (s)	14,59	8,98	10,60	12,77

construções. Particularmente, encontramos grandes diferenças em a melhoria do desvio de 10 a 100 construções, mas essa melhoria é reduzida à medida que aumentamos o número de construções. No entanto, o tempo aumenta consideravelmente. Para relacionar estes valores com a execução real do algoritmo é importante notar que uma rota de separação média pode durar de 250 a 1200s.

4.3.2 Bairros VND

O algoritmo VND proposto na Seção 3 inclui três bairros diferentes (veja Sect. 3.2.2), nomeado como N_1, N_2, e N_3 Uma das principais decisões com relação aos bairros incluídos em um VND é determinar a ordem em que eles são colocados dentro do algoritmo. Muitos pesquisadores definem a ordem da vizinhança dependendo do tamanho (vizinhanças menores vêm primeiro, pois são mais rápidas de atravessar). No entanto, consideramos colocar na primeira posição não apenas a vizinhança menor (N_3) mas também o segundo menor (N_2). Com isso em mãos, relatamos o desempenho das diferentes ordens possíveis do bairros dentro do VND (ver Tabela 8)

Como podemos ver na Tabela 8 a melhor combinação de bairros é N_3, N_1, e N_2, com desvio médio de 0,67%. No entanto, a combinação com o maior número de Os melhores valores são N_2, N_1, e N_3 Neste caso, decidimos usar a combinação com o menor desvio (N_3, N_1, e N_2) para nossa configuração final.

4.3.3 Contribuição de cada bairro

Um parâmetro chave ao projetar um VND é verificar se todas as vizinhanças incluídas no algoritmo contribuem para o processo de pesquisa. Caso algum bairro não apresente melhorias, ele deve ser removido e o tempo economizado utilizado para outras tarefas. Para testar esse fato, relatamos o número de melhorias feitas em cada bairro quando combinadas no método VND. Levando em consideração o conjunto de dados preliminares, fornecemos uma solução aleatória como ponto de partida para o VND. Na tabela9 relatamos o número de melhorias realizadas pela busca local que percorre cada bairro. Como mostra a tabela, todos os bairros produzem melhorias na busca. Particularmente,

Tabela 9 Melhorias produzidas por cada pesquisa local dentro do VND

	# melhorias	% de melhorias
N_1	24	35,82
N_2	26	38,81
N_3	17	25,37

Tabela 10 Comparação do VND com relação à pesquisa local

	VND	LS-1	LS-2	LS-3
Dev. (%)	0,12	5,42	8,34	10,21
# Melhor	15	1	0	0
CPUt (s)	22,89	4,03	13,83	1,13

N_1 e N_2 são os bairros que produzem o maior número de benfeitorias (24 e 26 respectivamente). No entanto, a diferença com relação a N_3 não é muito notável.

4.3.4 Comparação entre VND e procedimentos de pesquisa local

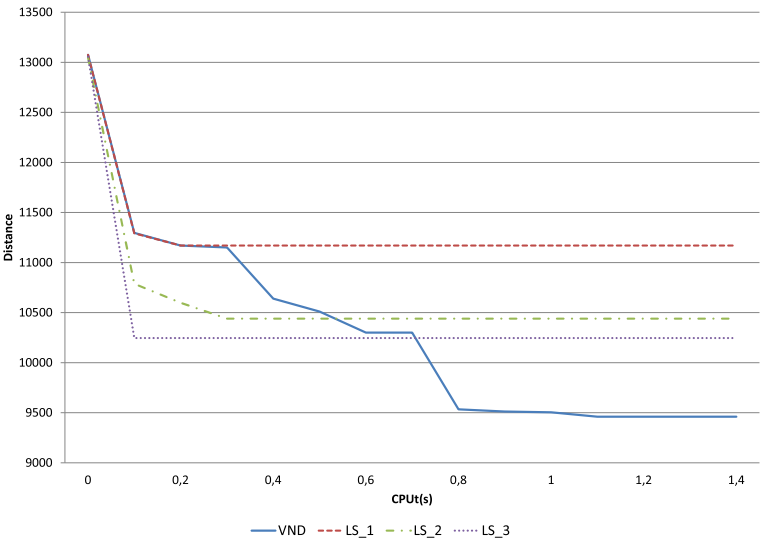
Um procedimento de busca local é executado até que nenhuma outra melhoria possa ser feita em uma vizinhança para uma função objetivo particular, ou seja, a solução obtida é um ótimo local. Por outro lado, a combinação de diferentes procedimentos de busca local dentro de um VND deve fornecer um ótimo local com respeito a todas as estruturas de vizinhança consideradas. Além disso, espera-se que a qualidade da solução encontrada por um procedimento VND seja melhor do que a solução encontrada por cada busca local isoladamente.

Na tabela 10 relatamos o desvio médio, o número das melhores soluções encontradas e o tempo de CPU de quatro algoritmos diferentes. Por um lado, executamos o procedimento VND partindo de uma solução aleatória, que foi fornecida como um parâmetro de entrada. Esta solução aleatória também foi fornecida a cada um dos procedimentos de busca local separadamente. Nesse caso, é possível notar que o procedimento VND é capaz de encontrar uma solução muito melhor do que cada busca local isoladamente. Porém, como era de se esperar, o tempo de CPU do VND é maior do que o tempo de CPU de cada busca local de forma independente.

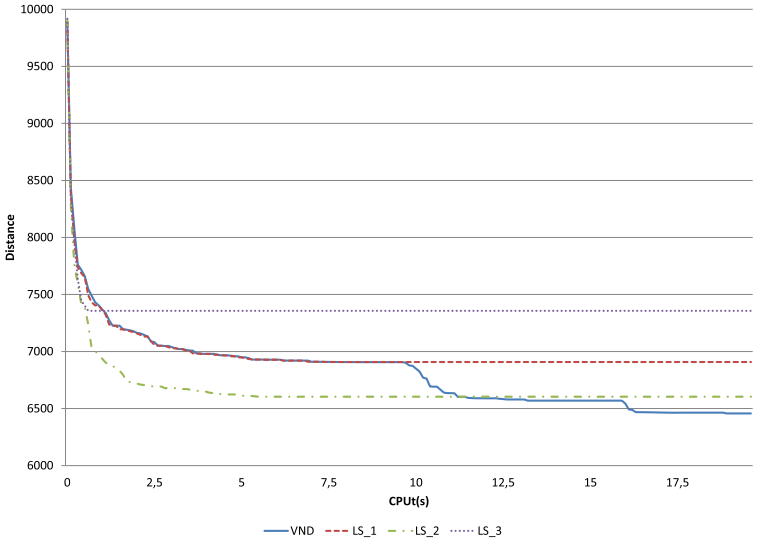
Para complementar os resultados apresentados na Tabela 10 apresentamos a evolução do valor da função objetivo ao longo do tempo de cada procedimento de busca local em comparação com o método VND. Neste experimento, selecionamos duas instâncias representativas do conjunto de dados preliminar: uma pequena (80 pedidos, representada na Fig.7a) e um grande (250 pedidos, representado na Fig. 7b). Conforme mostrado nas fi guras, o desempenho do VND é o melhor entre todos os métodos comparados, para as duas instâncias consideradas. No entanto, os procedimentos de busca local convergem para um ótimo local mais rapidamente do que o VND.

4.3.5 Desempenho da abordagem exata estática no estado da arte

Nosso último experimento preliminar é dedicado a ilustrar a adequação do uso de abordagens heurísticas no contexto da OOBP, em vez de usar uma abordagem exata. Para tanto, testamos o desempenho da formulação matemática introduzida na Seção2 sobre o conjunto de dados preliminares. É importante notar que o modelo matemático proposto no estado da arte foi projetado para a versão estática do problema (ou seja, todos os pedidos estão disponíveis no início do processo). Além disso, o modelo está incompleto e pode



uma Instância *C*5_*abc*2_80_46 composto por 80 pedidos.



b Instância *C*2_250_090 composto por 250 pedidos.

Fig. 7 Evolução dos procedimentos de Pesquisa Local e do método VND em duas instâncias particulares

não pode ser executado diretamente em um solver, pois a função “d” (que calcula a distância de uma determinada solução) não está definida no artigo original onde o modelo foi proposto. A fim de superar essa dificuldade, usamos a formulação da distância proposta em [31]

Executamos o modelo matemático para cada instância no Gurobi 9.0.0rc2 (win64). Como os tamanhos das instâncias consideradas são muito grandes para o modelo, reduzimos o tamanho de cada instância selecionando um número reduzido de pedidos (escolhidos aleatoriamente) de cada instância. Começamos com 5 pedidos por instância e aumentamos o número de pedidos por instância um

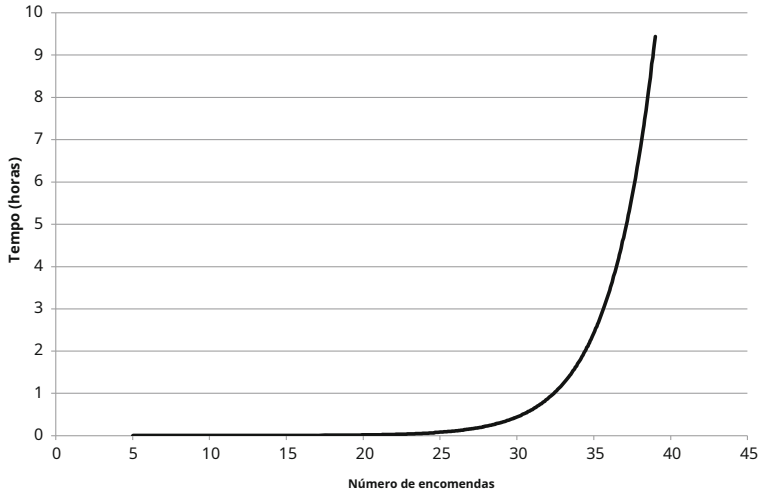


Fig. 8 Desempenho médio do matemático formulação introduzida em [18]

por um. Como esperado, o modelo matemático foi capaz de resolver instâncias de pequeno porte para a solução ótima. Porém, ao atingir os tamanhos de instância considerados neste artigo (maiores que 40 pedidos por instância), o tempo necessário excedeu o horizonte de tempo considerado.

Na Fig. 8relatamos a regressão não linear obtida ao considerar os resultados fornecidos pelo solver, que mostra a tendência de aumento do tempo quando o tamanho da instância também aumenta.

4.4 Experimentos finais

Uma vez que tenhamos testado os diferentes componentes de nossos algoritmos, configuramos nossa melhor variante do método GRASP + VND com os seguintes parâmetros: o número de iterações de GRASP + VND não é pré-definido. O procedimento é executado tantas vezes quanto possível entre a saída do selecionador para coletar um lote e a chegada do selecionador ao depósito após a coleta de todos os itens, mais o tempo de preparação. O α valor no procedimento construtivo GRASP foi definido como aleatório e ele é redefinido em cada iteração do método. Os três bairros proposto na Seção 3.2.2 foram incluídos no VND seguindo a ordem: N_3, N_1, e N_2

A melhor configuração do nosso algoritmo GRASP + VND (GR + VND nas tabelas) é em comparação com o melhor método no estado da arte, a Pesquisa Local Iterada (ILS) proposta em [18], e denotado nas experiências finais como ILS. Esta comparação final foi realizada nos dois conjuntos de dados apresentados anteriormente. Relatamos o resumo dos resultados em tabelas11 e 12.

Na tabela 11 consideramos a minimização do tempo máximo de conclusão como função objetivo. Esta função objetivo foi proposta em [18] e retorna o momento em que o último pedido processado é concluído. Em outras palavras, essa função objetivo é igual ao tempo total necessário para coletar todos os pedidos que chegaram ao depósito. Os resultados para a outra função objetivo considerada neste artigo são relatados na Tabela12. Nesse caso, a comparação com o estado da técnica considera a minimização do tempo máximo de giro de um pedido como função objetivo. Este é o tempo máximo que um pedido permanece no

Tabela 11 Comparação com o estado da arte considerando a minimização do tempo máximo de conclusão como função objetivo

	Albareda		Henn		Total	
	GR + VND	ILS [18]	GR + VND	ILS [18]	GR + VND	ILS [18]
Dev. (%)	0,10	3,38	0,11	2,41	0,11	2,90
# Melhor	60	4	55	11	115	15

Tabela 12 Comparação com o estado da arte considerando a minimização do tempo máximo de giro de uma ordem como função objetivo

	Albareda		Henn		Total	
	GR + VND	ILS [18]	GR + VND	ILS [18]	GR + VND	ILS [18]
Dev. (%)	0,29	8,14	0,63	5,80	0,46	6,97
# Melhor	61	3	52	12	113	15

sistema (ou seja, a diferença entre a hora de chegada ao armazém e a hora em que é concluído).

Apesar de fornecermos ambas as funções objetivo, consideramos que a mais interessante, atendendo a cenários reais para este problema, é o tempo de rotação (ver Tabela 12) Na verdade, nossos algoritmos foram configurados apenas para minimizar essa função objetivo (não o tempo máximo de conclusão). Relatamos apenas o tempo máximo de conclusão para as soluções encontradas ao minimizar o tempo máximo de giro. A principal razão de incluir o tempo máximo de conclusão é para fins de clareza, uma vez que esta função objetivo foi a principal função objetivo proposta em [18]

Considerando a minimização do tempo máximo de conclusão relatado na Tabela 11, nós aprecio que GRASP + VND é o melhor método geral, uma vez que atinge 0,11% de desvio em relação aos melhores valores encontrados no experimento, enquanto ILS atinge 2,90% de desvio. Além disso, o número de melhores soluções encontrado por GRASP + VND (115) é muito maior do que o número de melhores soluções encontrado por ILS (15).

No que diz respeito à minimização do tempo máximo de giro, relatamos os valores obtidos por cada algoritmo na Tabela 12. Novamente, GRASP + VND parece ser o melhor método. Nesse caso, o desvio obtido (0,46%) é bem melhor que o do ILS (6,97%). O número de melhores soluções encontradas por GRASP + VND (113) também é muito maior do que o número de melhores soluções encontradas por ILS (15).

Para completar a comparação dos resultados, foi realizado um teste de diferença média dos resultados obtidos pelos dois métodos. Mesmo quando a hipótese de normalidade não é verificada, o tamanho da amostra usada (128 no total, 64 da Albareda e 64 de Henn) nos permite usar o teste t para a diferença de médias em amostras pareadas (observe que ambos os métodos foram testado nas mesmas instâncias). O resultado não poderia ser mais significativo: com um p-valor menos do que 10^{-10}, a hipótese nula H_0: $\mu_{ILS} \leq \mu_{GV}$ pode ser rejeitado, onde μ_{EuLP} e μ_{GV} são temáticos da função objetivo (rotatividade ou distância) por ILS e GRASP + VND, respectivamente. Portanto, pode-se aceitar que os resultados propostos pelo GRASP + VND são estatisticamente melhores do que os fornecidos pelo ILS. Realizamos esses testes para os dois conjuntos de instâncias (Albareda e Henn) separadamente, e os resultados são semelhantes, com p-valores praticamente nulos em todos os casos.

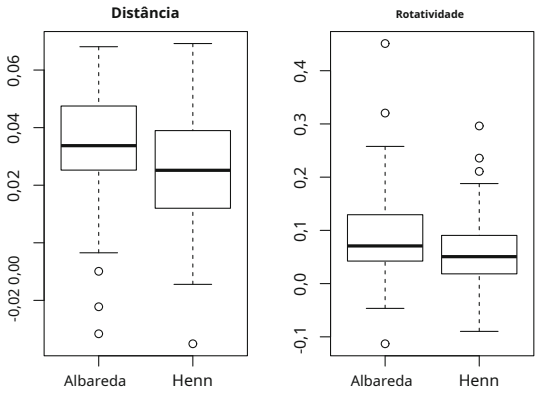


Fig. 9 Diferença relativa entre os objetivos ILS e GRASP + VND

Para concluir, analisamos a melhoria relativa do GRASP + VNDversus ILS. Figura9 mostra a distribuição da proporção $z_{ILS}-z_{GVNS}$ para os dois objetivos e os dois conjuntos de instâncias, Albareda e Henn. Os dois boxplots à direita mostram os resultados para o volume de negócios e as duas à esquerda para a distância. Para cada um dos objetivos, a melhoria relativa é mostrada separadamente para as instâncias Albareda e Henn. Observe que valores negativos implicam que ILS fornece melhores resultados do que GRASP + VND. A partir desses boxplots, podemos concluir novamente que, na maioria dos casos, GRASP + VND oferece soluções melhores; nos poucos casos em que a solução ILS é melhor, não excede 2% de melhoria em comparação com quase 7% que GRASP + VND pode melhorar a solução. É interessante notar que GRASP + VNDachieves menos melhorias nas instâncias de Henn, que são precisamente onde o ILS foi testado.

Para facilitar futuras comparações, relatamos os melhores valores encontrados em nossos experimentos, para cada uma das instâncias consideradas. Particularmente, na Tabela13 no Apêndice A, relatamos os resultados para o conjunto de dados introduzido em [1], e os resultados para o conjunto de dados introduzido em [18] na tabela 14 no Apêndice B. Todas essas informações, juntamente com o conjunto de instâncias usadas em nossos experimentos, estão disponíveis em http://grafo.etsii.urjc.es/optsicom/oobp/.

5. Conclusões

Neste artigo, abordamos o problema de lote de pedidos on-line com um único coletor e em um depósito de bloco único. Este problema é uma variante da conhecida família de problemas comumente referida como Order Batching Problem. Baseia-se na estratégia de separação por lote de pedidos como uma forma eficiente de recuperar os pedidos recebidos em um depósito, ou seja, os pedidos são agrupados em lotes antes de serem coletados. Além disso, cada lote é coletado em uma única rota de um selecionador.

O OOBP apresenta a dificuldade de que todos os pedidos que devem ser recuperados não estejam disponíveis no início do processo, mas cheguem ao depósito a qualquer momento. Por uma questão de simplicidade, consideramos um horizonte de tempo de 4h para a chegada dos pedidos, entretanto, o selecionador pode precisar de tempos maiores para coletar todos os pedidos chegados naquele horizonte de tempo.

Para atingir nossos objetivos, propusemos vários algoritmos heurísticos que são combinados dentro de uma metaheurística GRASP e VND. Particularmente, propomos um algoritmo GRASP como uma estrutura geral para lidar com o problema, e usamos a metaheurística VND como um procedimento de busca local. A fase construtiva do GRASP é baseada no peso dos pedidos como um critério guloso (ou seja, o mais pesado, o primeiro). O α parâmetro de GRASP foi

definido como um valor aleatório em cada iteração. O método VND proposto inclui três estruturas de vizinhança diferentes que foram classificadas empiricamente. Por fim, utilizamos o algoritmo S-Shape como estratégia de roteamento para calcular a melhor rota de coleta dos pedidos agrupados em cada lote.

Vale ressaltar que neste trabalho consideramos duas funções objetivo distintas: (i) minimizar o tempo máximo de conclusão; e (ii) minimizar o tempo máximo de giro. O primeiro minimiza o tempo que leva para coletar todos os pedidos que chegam ao sistema. Esta função objetivo é amplamente utilizada em muitos artigos. Este último minimiza o tempo máximo que um pedido permanece no sistema. Até onde entendemos o problema, esta função objetivo representa o cenário mais realista para esta variante do OBP. No entanto, reportamos e comparamos com propostas anteriores no estado da arte, os valores obtidos para ambas as funções objetivo. Os algoritmos propostos melhoram os resultados anteriores no estado da arte em ambos os casos, encontrando melhorias de mais de 3% e 6%, em média, para cada função objetivo, respectivamente.

Nossos resultados indicaram que o tempo de execução para este problema não é um dos maiores problemas, uma vez que o horizonte de tempo em um cenário real é muito grande. Existem também muitos pequenos fatores que devem ser levados em consideração, como o horário de saída do coletor, uma vez que os lotes estão prontos para serem coletados. Em uma primeira abordagem, quanto mais cedo melhor, entretanto, aguardar pode resultar em uma melhoria final, uma vez que há mais chances de chegada de novos pedidos ao sistema, que podem ser incluídos no próximo lote. Além disso, descobrimos que o uso de vários bairros é uma estratégia fundamental, uma vez que nem sempre é fácil realizar movimentos dentro do espaço disponível nos lotes. Detectamos que classificar os pedidos em um peso decrescente pode ajudar a obter lotes com menos espaço desperdiçado. Os resultados empíricos indicam que quanto menor o número de lotes,

É importante destacar que ao considerar o tempo total necessário para a coleta de todos os pedidos como função objetivo, o tempo de coleta de um determinado lote não muda, não importando o momento em que se inicia a rota de coleta desse lote. Porém, ao considerar o tempo de giro como função objetivo, não só a configuração do lote é importante, mas também o momento em que começa a rota de separação para cada lote. Essa é uma dificuldade adicional dessa variante do problema. Nesse sentido, é necessário definir um momento no tempo como ponto de partida hipotético, tomado como referência, para comparar duas soluções diferentes.

Como observação final, a versão online do OBP possibilita muitas situações em que um pequeno número de pedidos é tratado ao mesmo tempo. Portanto, pode ser uma oportunidade interessante para projetar algoritmos matemáticos no futuro.

Apêndice A: Resultados por instância (conjunto de instâncias Albareda [1])

Tabela 13 Resultados por instância para o conjunto de dados introduzido em [1] sobre as duas funções objetivo consideradas

Instância	Função objetiva	Instância	Função objetiva
	T (s) Volume de negócios (s)		T (s) Volume de negócios (s)
W1_100_000	22, 308 10, 864	W3_100_000	39, 902 30, 959
W1_100_030	18.371 6394	W3_100_030	33, 212 25, 539
W1_100_060	21, 676 9911	W3_100_060	41.067 33.276
W1_100_090	18, 313 6157	W3_100_090	33, 974 24, 111
W1_150_000	28, 612 18, 449	W3_150_000	54, 189 45, 495
W1_150_030	24, 386 13, 556	W3_150_030	46, 013 38, 616
W1_150_060	30, 770 18, 812	W3_150_060	58, 797 50, 833
W1_150_090	24, 101 12, 129	W3_150_090	46, 210 38, 277
W1_200_000	42.091 29.194	W3_200_000	72, 452 63, 494
W1_200_030	30, 810 18, 461	W3_200_030	60, 713 54, 808
W1_200_060	38, 485 27, 209	W3_200_060	79, 633 72, 394
W1_200_090	31, 405 20, 774	W3_200_090	57.516 52.072
W1_250_000	53, 101 40, 683	W3_250_000	93, 604 84, 912
W1_250_030	38, 490 26, 414	W3_250_030	74, 203 68, 954
W1_250_060	51, 255 40, 891	W3_250_060	99, 265 91, 559
W1_250_090	40, 606 30, 471	W3_250_090	76, 361 71, 208
W2_100_000	17, 203 6153	W4_100_000	110, 218 96, 620
W2_100_030	15, 535 2702	W4_100_030	92, 936 79, 951
W2_100_060	17, 131 4849	W4_100_060	94, 370 82, 276
W2_100_090	15.069 2849	W4_100_090	77,919 64,560
W2_150_000	24, 227 13, 214	W4_150_000	155,919 14,1927
W2_150_030	21.052.9595	W4_150_030	118, 893 105, 478
W2_150_060	24, 375 13, 561	W4_150_060	155,998 141,955
W2_150_090	21, 183 10, 110	W4_150_090	119, 539 106, 666
W2_200_000	33, 170 22, 498	W4_200_000	198, 530 186, 439
W2_200_030	26, 341 15, 274	W4_200_030	150, 094 136, 789
W2_200_060	31.020 21.247	W4_200_060	202, 348 188, 713
W2_200_090	26, 429 15, 433	W4_200_090	169, 074 154, 963
W2_250_000	38, 100 28, 017	W4_250_000	249, 690 236, 033
W2_250_030	33, 341 21, 956	W4_250_030	181, 508 168, 829
W2_250_060	41, 148 30, 372	W4_250_060	249, 863 235, 831
W2_250_090	34, 352 23, 382	W4_250_090	199, 738 185, 948

Apêndice B: Resultados por instância (conjunto de instâncias Henn [18])

Tabela 14 Resultados por instância para o conjunto de dados introduzido em [18] sobre os dois funções objetivas consideradas

Instância	Função objetiva	Instância	Função objetiva
	T (s) Volume de negócios (s)		T (s) Volume de negócios (s)
W5_abc1_40_29	21, 109 9848	W5_ran1_40_29	24, 689 12, 838
W5_abc1_40_30	17, 541 5133	W5_ran1_40_30	18.023 8715
W5_abc1_40_31	17, 831 4788	W5_ran1_40_31	18.815.6.364
W5_abc1_40_32	16, 226 4488	W5_ran1_40_32	16.393.598
W5_abc1_60_37	31, 794 19, 729	W5_ran1_60_37	36, 681 25, 558
W5_abc1_60_38	23, 366 10, 978	W5_ran1_60_38	26, 418 14, 984
W5_abc1_60_39	21.061.10.985	W5_ran1_60_39	23, 674 12, 124
W5_abc1_60_40	18, 196 8538	W5_ran1_60_40	20, 759 9480
W5_abc1_80_61	40, 745 27, 372	W5_ran1_80_61	47, 254 34, 077
W5_abc1_80_62	32, 385 20, 458	W5_ran1_80_62	37, 688 24, 958
W5_abc1_80_63	26.083 14.713	W5_ran1_80_63	29, 133 17, 544
W5_abc1_80_64	24, 189 12, 530	W5_ran1_80_64	27, 565 15, 419
W5_abc1_100_69	45, 613 32, 404	W5_ran1_100_69	53.976.41.064
W5_abc1_100_70	33, 888 21, 849	W5_ran1_100_70	39, 367 27, 710
W5_abc1_100_71	30.081 20.595	W5_ran1_100_71	34.027 23.168
W5_abc1_100_72	28, 543 17, 174	W5_ran1_100_72	31, 446 19, 298
W5_abc2_40_9	21, 729 9279	W5_ran2_40_9	24, 660 14, 619
W5_abc2_40_10	18, 170 9377	W5_ran2_40_10	21.073.8990
W5_abc2_40_11	15, 708 4829	W5_ran2_40_11	16.051.5874
W5_abc2_40_12	15, 129 4177	W5_ran2_40_12	16, 623 5166
W5_abc2_60_17	29, 225 17, 812	W5_ran2_60_17	34, 631 22, 110
W5_abc2_60_18	25, 303 12, 760	W5_ran2_60_18	27, 725 17, 760
W5_abc2_60_19	20, 510 8120	W5_ran2_60_19	23, 565 13, 142
W5_abc2_60_20	18, 304 8467	W5_ran2_60_20	19, 750 8963
W5_abc2_80_45	38.013 24.949	W5_ran2_80_45	44, 605 31, 155
W5_abc2_80_46	28, 691 16, 423	W5_ran2_80_46	32, 758 21, 503
W5_abc2_80_47	25, 956 14, 817	W5_ran2_80_47	30, 911 18, 730
W5_abc2_80_48	23.524 13.018	W5_ran2_80_48	26, 983 16, 964
W5_abc2_100_53	49, 363 36, 092	W5_ran2_100_53	57.097 43.018
W5_abc2_100_54	35.046 24.269	W5_ran2_100_54	40.093.27.849
W5_abc2_100_55	34, 670 23, 567	W5_ran2_100_55	39, 278 28, 689
W5_abc2_100_56	28, 941 18, 828	W5_ran2_100_56	32.025 20.591

Referências

1. Albareda-Sambola, M., Alonso-Ayuso, A., Molina, E., De Blas, CS: Pesquisa de vizinhança variável para lote de pedidos em um armazém. Asia-Pac. J. Oper. Res.**26 (**5), 655-683 (2009)
2. Chen, F., Wei, Y., Wang, H .: Um método de atribuição e lote baseado em heurística para pedidos de clientes online. Flex. Serv. Manuf. J.**30 (**4), 640-685 (2018)
3. Coyle, JJ, Bardi, EJ, Langley, CJ, et al .: TheManagement ofBusinessLogistics, vol. 6.West Publishing Company Minneapolis, St Paul (1996)
4. De Koster, MBM, Van der Poort, ES, Wolters, M .: E fi cient orderbatching methods in warehouses. Int. J. Prod. Res.**37 (**7), 1479-1504 (1999)
5. De Koster, R., Le-Duc, T., Roodbergen, KJ: Design and control of warehouse order picking: a literatura review. EUR. J. Oper. Res.**182,** 481–501 (2007)
6. De Koster, R., Jan Roodbergen, K., van Voorden, R .: Redução do tempo de caminhada no centro de distribuição de de bijenkorf. In: Novas tendências em logística de distribuição, pp. 215–234. Springer, Berlim (1999)
7. Drury, J., Grupo de Interesse Especial de Operações de Armazém, Grupo de Trabalho de Seleção de Pedidos, Turnbull, B., Institute of Logistics (Great Britain): Towards More Eficient Order Picking. Monografia do IMM. Instituto de Gestão de Materiais. ISBN 9781870214063 (1988).https://books.google.es/books? id = LbTKAAAACAAJ
8. Duarte, A., Pantrigo, JJ, Pardo, EG, Mladenovic, N .: Pesquisa de vizinhança de variáveis multi-objetivo: uma aplicação a problemas de otimização combinatória. J. Global Optim.**63 (**3), 515-536 (2015)
9. Duarte, A., Pantrigo, JJ, Pardo, EG, Sánchez-Oro, J .: Estratégias de busca de vizinhança de variável paralela para o problema de minimização de largura de corte. IMA J. Manag. Matemática.**27 (**1), 55-73 (2013)
10. Feo, TA, Resende, M .: Greedy randomized randomized search procedures. J. Global Optim.**6,** 109-133 (1995)
11. Gademann, AJRM, Van Den Berg, JP, Van Der Hoff, HH: Um algoritmo de envio em lote para separação em onda em um armazém de corredor paralelo. IIE Trans.**33 (**5), 385-398 (2001)
12. Gademann, N., Velde, S .: Faça pedidos em lotes para minimizar o tempo total de viagem em um depósito de corredor paralelo. IIE Trans.**37 (**1), 63-75 (2005)
13. Gibson, DR, Sharp, GP: Procedimentos de envio em lote. EUR. J. Oper. Res.**58 (**1), 57-67 (1992)
14. Gu, J., Goetschalckx, M., McGinnis, LF: Pesquisa em design de armazém e avaliação de desempenho: uma revisão abrangente. EUR. J. Oper. Res.**203 (**3), 539-549 (2010)
15. Hall, RW: Aproximações de distância para roteamento de selecionadores manuais em um depósito. IIE Trans.**25 (**4), 76-87 (1993)
16. Hansen, P., Mladenović, N .: Pesquisa de vizinhança variável: princípios e aplicações. EUR. J. Oper. Res. **130 (**3), 449-467 (2001)
17. Hansen, P., Mladenović, N., Moreno-Pérez, JA: Pesquisa de vizinhança variável: métodos e aplicações. Ann. Oper. Res.**175 (**1), 367-407 (2010)
18. Henn, S .: Algoritmos para lote de pedidos on-line em um armazém de separação de pedidos. Comput. Oper. Res. **39 (**11), 2549–2563 (2012)
19. Henn, S., Koch, S., Doerner, K., Strauss, C., Wäscher, G .: Metaheuristics for the order batching problem in manual order picking systems. Ônibus BuR. Res. J.**3 (**1), 82-105 (2010)
20. Henn, S., Schmid, V .: Metaheuristics para lote de pedidos e sequenciamento em sistemas de separação de pedidos manuais. Comput. Ind. Eng.**66 (**2), 338-351 (2013)
21. Henn, S., Wäscher, G .: Heurísticas de pesquisa tabu para o problema de lote de pedidos em sistemas de separação de pedidos manuais. EUR. J. Oper. Res.**222 (**3), 484-494 (2012)
22. Ho, Y.-C., Tseng, Y.-Y .: Um estudo sobre os métodos de coleta de pedidos em lote em um centro de distribuição com dois corredores cruzados. Int. J. Prod. Res.**44 (**17), 3391-3417 (2006)
23. Hsu, C.-M., Chen, K.-Y., Chen, M.-C .: Batching orders in warehouses, minimizando a distância de viagem com algoritmos genéticos. Comput. Ind.**56 (**2), 169-178 (2005)
24. Koch, S., Wäscher, G .: Um algoritmo genético de agrupamento para o Order Batching Problem in Distribution Warehouses. J. Bus. Econ.**86 (**1-2), 131-153 (2016)
25. Menéndez, B., Bustillo, M., Pardo, EG, Duarte, A .: Pesquisa de vizinhança de variável geral para o problema de lote e sequenciamento de pedidos. EUR. J. Oper. Res.**263 (**1), 82-93 (2017)
26. Menéndez, B., Pardo, EG, Alonso-Ayuso, A., Molina, E., Duarte, A .: Estratégias de busca de vizinhança variável para o problema de batching de ordens. Comput. Oper. Res.**78,** 500-512 (2017)
27. Menéndez, B., Pardo, EG, Duarte, A., Alonso-Ayuso, A., Molina, E .: Pesquisa de vizinhança de variável geral aplicada ao processo de picking em um armazém. Elétron. Notes Discrete Math.**47,** 77–84 (2015)
28. Menéndez, B., Pardo, EG, Sánchez-Oro, J., Duarte, A .: Pesquisa de vizinhança de variável paralela para o problema de lote de ordens mín-máx. Int. Trans. Oper. Res,**24 (**3), 635-662 (2017)
29. Mladenović, N., Hansen, P .: Pesquisa de vizinhança variável. Comput. Oper. Res.**24 (**11), 1097-1100 (1997)

30. Öncan, T .: Formulações MILP e um algoritmo de busca local iterado com limiar de Tabu para o problema de lote de pedidos. J. Oper. Res.**243,** 142-155 (2015)
31. Öncan, T., Cağırıcı, M .: Formulações MILP para o problema de lote de pedidos em sistemas de armazenamento de peças separadas. IFAC Proc. Vol.**46 (**9), 471-476 (2013)
32. Pan, C.-H., Liu, S.-Y .: A comparative study of order batching algorítmos. Ómega**23 (**6), 691-700 (1995)
33. Pardo, EG, Mladenović, N., Pantrigo, JJ, Duarte, A .: Pesquisa de formulação variável para o problema de minimização de largura de corte. Appl. Soft Comput.**13 (**5), 2242–2252 (2013)
34. Pérez-Rodríguez, R., Hernández-Aguirre, A., Jöns, S .: Uma estimativa contínua do algoritmo de distribuição para o problema de lote de pedidos on-line. Int. J. Adv. Manuf. Technol.**79 (**1), 569–588 (2015)
35. Petersen, CG: Uma avaliação das políticas de roteamento de separação de pedidos. Int. J. Oper. Prod. Manag.**17 (**11), 1098-1111 (1997)
36. Petersen, CG: Roteamento e interação da política de armazenamento nas operações de separação de pedidos Decis. Sci. Inst. Proc. **31 (**3), 1614-1616 (1995)
37. Ratliff, HD, Rosenthal, AS: Seleção de pedidos em um armazém retangular: um caso solucionável do problema do caixeiro viajante. Oper. Res.**31 (**3), 507-521 (1983)
38. Roodbergen, KJ, Koster, RD: Métodos de roteamento para armazéns com vários corredores transversais. Int. J. Prod. Res.**39 (**9), 1865-1883 (2001)
39. Roodbergen, KJ, Petersen, CG: Como melhorar a eficiência da separação de pedidos com políticas de roteamento e armazenamento. Em: GR Forger et al. (eds.), Progress inMaterial Handling Practice. Material Handling Institute, Charlotte, North Carolina, pp. 107-124 (1999)
40. Rosenwein, MB: Uma comparação de heurísticas para o problema de pedidos em lote para seleção de armazém. Int. J. Prod. Res.**34 (**3), 657-664 (1996)
41. Rubrico, JIU, Higashi, T., Tamura, H., Ota, J .: Reprogramação online de vários agentes de separação para gerenciamento de depósito. Robô. Comput. Integr. Manuf.**27 (**1), 62-71 (2011)
42. Scholz, A., Schubert, D., Wäscher, G .: Separação de pedidos com vários separadores e datas de vencimento - solução simultânea de lotes de pedidos, atribuição e sequenciamento de lotes e problemas de roteamento de separadores. EUR. J. Oper. Res.**263 (** 2), 461-478 (2017)
43. Scholz, A., Wäscher, G .: Order Batching and Picker Routing em sistemas de separação de pedidos manuais: os benefícios do roteamento integrado. CEJOR**25 (**2), 491-520 (2017)
44. Tang, LC, Chew, EP: Sistemas de separação de pedidos: estratégias de atribuição de lotes e armazenamento. Comput. Ind. Eng.**33 (**3), 817–820 (1997). Artigos selecionados dos anais de 1996 ICC & IC
45. VanNieuwenhuyse, I., Colpaert, J .: Encomende lote em armazéns de coleta e classificação de vários servidores. DTEW-KBI **0731,** 1-29 (2007)
46. Zhang, J., Wang, X., Chan, FTS, Ruan, J .: On-line order batching and sequencing problemwithmultiple pickers: a Hybrid rule-based algorithm. Appl. Matemática. Modelo.**45,** 271-284 (2017)
47. Zhang, J., Wang, X., Huang, K .: Programação on-line integrada de lote de pedidos e entrega sob comércio eletrônico b2c. Comput. Ind. Eng.**94,** 280–289 (2016)
48. Zhang, J., Wang, X., Huang, K .: Programação on-line de coleta e entrega de pedidos com várias zonas e capacidade limitada de veículos. Ómega**79,** 104-115 (2018)
49. Žulj, I., Kramer, S., Schneider, M .: Um híbrido de busca adaptativa de grande vizinhança e busca tabu para o problema de lote de pedidos. EUR. J. Oper. Res.**264 (**2), 653-664 (2018)

A ffi liações

Sergio Gil-Borrás1 · Eduardo G. Pardo2 · Antonio Alonso-Ayuso2 · Abraham Duarte2

Sergio Gil-Borrás
sergio.gil@upm.es

Antonio Alonso-Ayuso
antonio.alonso@urjc.es

Abraham Duarte
abraham.duarte@urjc.es

1 Universidad Politécnica de Madrid, C / Alan Turing s / n (Ctra. De Valencia, Km. 7), 28031 Madrid, Espanha

2 Universidad Rey Juan Carlos, C / Tulipán s / n, 28933 Móstoles, Madrid, Espanha