Optimization Approaches in Order Assignment

Alejandro Cassis[1][0009-0005-6081-0368], Martin Gross[2][0000-0001-9220-5289], Nikos Liakopoulos[2][0000-0002-9339-8921], Georgios Paschos[2][0000-0002-5922-1612], Christian Schulz[1(✉)⋆][0000-0002-2823-3506], and Alexander Svozil[3][0000-0002-2389-4874]

[1] Amazon Research, Germany
{alecaar,chrsschz}@amazon.de
[2] Amazon Research, Luxembourg
{grosmar,nliako,paschos}@amazon.lu
[3] Amazon Research, USA
asvozil@amazon.com

Abstract. We study the problem of assigning customer orders to fulfillment centers in Amazon's logistics network. We analyze this assignment problem in both offline and online settings, developing a range of algorithms for each scenario. In the offline setting, we present solver-based and heuristic approaches including mixed integer programming (MIP), best-fit algorithms, and local search methods. For the online case, where orders must be assigned as they arrive, we develop controllers ranging from simple greedy strategies to approaches based on capacity-aware surcharges and periodic reoptimization. Our experimental results show that consumption-scaled approaches outperform cost-based methods in oversubscribed settings, local search provides scalable improvements while remaining amenable to parallelization, batch processing matches a 300-second MIP at reduced computational cost, and periodic reoptimization consistently improves online performance. These findings provide practical insights for our production systems.

Keywords: Order assignment · logistics optimization · online algorithms

1 Introduction

We study Amazon's order assignment problem where customer orders must be assigned to fulfillment resources. At each time epoch, a customer order appears that needs to be served using available resources. The controller, i.e. an algorithm, needs to decide which resource, e.g. a fulfillment center, will be used from a compatible subset. Each resource has a usage cost that is particular to the type (or commodity) of the order, and there is an overall capacity for each resource. The goal is to minimize total cost of assignments.

⋆ Corresponding author: Christian Schulz, chrsschz@amazon.de.

The controller needs to operate very fast, serving millions of orders per hour, and therefore a tempting solution is to treat each order in a greedy manner: rank the compatible resources by cost for this order, and then simply assign the order to the cheapest option. Whenever a resource capacity is exhausted, it gets removed from these lists and can not be used any more. The problem with this greedy approach is that as we approach the end of the horizon and resource capacities become exhausted, the assignments may become suboptimal since we may be forced using very expensive assignments. Furthermore, it may be that an order has no feasible assignment any more, in which case we need to pay a high penalty M to serve this order through an alternative path outside Amazon. There would be cases where we would prefer to assign an order to the 2nd or 3rd ranked option, thereby saving capacity at the best resource for future orders.

Contribution. We provide an experimental evaluation of hybrid optimization approaches for order assignment that identifies four key principles for efficient systems at scale: (1) consumption-scaled approaches outperform pure cost-based methods in oversubscribed settings, demonstrating the importance of considering resource consumption in assignment decisions; (2) local search provides scalable solution improvements while remaining amenable to parallelization; (3) batch processing with limited-time mixed integer programming (MIP) solves matches a 300-second MIP at a fraction of the computational cost, making sophisticated optimization practical for larger instances; and (4) periodic reoptimization consistently outperforms traditional online methods, with batch algorithms transferring effectively to online settings. *Organization.* Section 2 defines the optimization problems. Section 3 reviews related work. Sections 4 and 5 present our offline and online algorithms, respectively. Section 6 provides the experimental evaluation, and Section 7 concludes with a summary and directions for future work.

2 Preliminaries

Basic Concepts. We define $\mathcal{O}$ to be the (in the online setting possibly infinite) set of orders, indexed by $o = 1, 2, \ldots$; each order has a nominal consumption (size) $w_o > 0$ and a target delivery station (DS) $d_o \in \mathcal{D}$; $\mathbf{d}_{od}$ is an indicator variable indicating that order o belongs to delivery station d; further let $\mathcal{F}$ be the set of fulfillment centers (FCs), indexed by f; there is also a virtual fulfillment center $\mathcal{V}$ to model third-party carriers. We assume indices for the different objects we defined are unique to simplify the following notation. For each fulfillment center $f \in \mathcal{F}$, $\mathbf{c}_f$ represents its capacity. For each delivery station $d \in \mathcal{D}$, $\mathbf{c}_d$ represents its capacity. There are links between facilities and delivery stations. Each link $e \in \mathcal{F} \times \mathcal{D}$ also has a capacity which we call $\mathbf{c}_e$. Moreover, an order o can only be processed by a subset of $\mathcal{F}_o \subseteq \mathcal{F}$ of the facilities – its reachable facilities. Processing an order o through fulfillment center $f \in \mathcal{F}_o$ has an associated per-unit cost γ_{of}, so that the assignment costs $\gamma_{of}\, w_o$; the virtual fulfillment center does not have capacity limits and if it is used, then no link or delivery station capacities are used. Costs for the virtual fulfillment center are set to a large

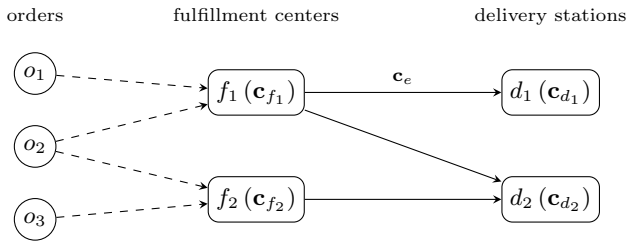


Fig. 1. Problem structure: each order o is assigned to one reachable fulfillment center (dashed), consuming w_{of} at that FC, w_{oe} at the link $e = (f, d_o)$, and w_{od} at its fixed delivery station d_o. Capacities $\mathbf{c}_f$, $\mathbf{c}_e$, $\mathbf{c}_d$ bound total consumption at fulfillment centers, links, and delivery stations, respectively.

penalty M, i.e. $\gamma_{o\mathcal{V}} = M$ and $w_{o\mathcal{V}} = w_o$. The parameter w_{or} denotes the resource consumption rate of order $o \in \mathcal{O}$ when assigned to resource r, where r can be a fulfillment center $r \in \mathcal{F}$, a link, or a delivery station $r \in \mathcal{D}$. We write w_{of}, w_{oe}, w_{od} for the consumption of order o at a fulfillment center f, a link e, and a delivery station d, which in general may differ across resources. The algorithms in this paper operate on the single per-order rate w_o, i.e. they assume $w_{of} = w_{oe} = w_{od} = w_o$. Figure 1 illustrates these entities and the three capacity types.

2.1 Offline Optimization Problem

The offline optimization problem assumes complete knowledge of all orders beforehand, so that $\mathcal{O} = \{1, \ldots, |\mathcal{O}|\}$ is finite, and minimizes the total assignment cost while respecting resource capacity constraints. When fulfillment center constraints prevent an order from being assigned internally, it must be redirected to a third-party provider at a penalty cost M. We introduce decision variables x_{of} which are 1 if order o has been assigned to fulfillment center f and zero otherwise. We then arrive at the following optimization problem:

$$\begin{aligned}
\underset{x}{\text{minimize}} \quad & \sum_{o\in\mathcal{O}} \sum_{f\in\mathcal{F}\cup\{\mathcal{V}\}} \gamma_{of} w_o x_{of} \\
\text{subject to} \quad & \sum_{f\in\mathcal{F}\cup\{\mathcal{V}\}} x_{of} = 1 \quad \forall o \in \mathcal{O} && \text{(exactly one assignment, incl. virtual FC } \mathcal{V}) \\
& \sum_{o\in\mathcal{O}} w_{of} x_{of} \le \mathbf{c}_f \quad \forall f \in \mathcal{F} && \text{(fulfillment center capacity)} \\
& \sum_{o\in\mathcal{O}} (\sum_{f\in\mathcal{F}} x_{of}) w_{od} \mathbf{d}_{od} \le \mathbf{c}_d \quad \forall d \in \mathcal{D} && \text{(delivery station capacity)} \\
& \sum_{o\in\mathcal{O}} w_{oe} \mathbf{d}_{od} x_{of} \le \mathbf{c}_e \quad \forall e = (f, d) \in \mathcal{F} \times \mathcal{D} && \text{(link capacity)} \\
& x_{of} = 0 \quad \forall o \in \mathcal{O}, f \in \mathcal{F} \setminus \mathcal{F}_o && \text{(compatibility)} \\
& x_{of} \in \{0, 1\} \quad \forall o \in \mathcal{O}, f \in \mathcal{F} \cup \{\mathcal{V}\}
\end{aligned}$$

The objective minimizes total assignment cost. The first constraint ensures each order is assigned to exactly one fulfillment center, including possibly the virtual one. The second and third constraints enforce capacity limits at fulfillment centers and delivery stations, respectively, by bounding the total consumption of assigned orders. The fourth constraint ensures that the flow on each link between a facility and a delivery station respects its capacity. The fifth constraint enforces compatibility, and the last constraint ensures binary assignments. Since each order has a single fixed delivery station d_o and, by the first constraint, exactly one fulfillment center, it consumes w_{of}, w_{oe}, and w_{od} once at the chosen center, its link, and its delivery station, respectively; orders are never split across resources.

2.2 Hardness and Novelty

Our offline problem generalizes the generalized assignment problem (GAP), which is NP-hard [15]. We reduce from the minimization variant of GAP, in which every item must be assigned to exactly one agent subject to per-agent capacities so as to minimize total cost, matching our setting where every order must be served (possibly by the virtual fulfillment center). Formally, GAP reads:

$$
\begin{aligned}
\text{minimize} \quad & \sum_{i \in M} \sum_{j \in N} c_{ij} x_{ij} \\
\text{subject to} \quad & \sum_{j \in N} x_{ij} = 1 && i \in M \quad \text{(Assignment)} \\
& \sum_{i \in M} w_{ij} x_{ij} \leq b_j && j \in N \quad \text{(Knapsack)} \\
& x_{ij} \in \{0,1\} && i \in M,\, j \in N
\end{aligned}
$$

An instance of GAP can be reduced to an instance of our offline problem as follows: set $\mathcal{O}$ to M, $\mathcal{F}$ to N. The fulfillment-center capacity of $j \in \mathcal{F}$ is set to b_j, the fulfillment-center consumption to $w_{if} = w_{ij}$ (GAP is NP-hard even for strictly positive weights, so we may take $w_{ij} > 0$), the nominal consumption to $w_i = 1$, and the cost to $\gamma_{ij} = c_{ij}$; the cost for the virtual fulfillment center is set to a sufficiently large finite value (e.g., exceeding $\sum_{i,j} c_{ij}$) so that it is used only when no feasible assignment exists. Moreover, $\mathcal{F}_i$ is set to $\mathcal{F}$, and capacities on links and delivery stations are set to ∞ so that they are never a bottleneck. The objective then reduces to $\sum_{i,j} c_{ij} x_{ij}$ and the fulfillment-center constraints become the GAP knapsack constraints, so an optimal solution to our problem yields an optimal GAP assignment and our problem is NP-hard. The single-rate variant used in our experiments ($w_{of} = w_{oe} = w_{od} = w_o$) is likewise NP-hard, as it already contains the multiple knapsack problem.

Our full problem adds two resource types beyond GAP (delivery-station and link capacities) that couple the assignments of fulfillment centers sharing a delivery station.

2.3 Online Optimization Problem

Unlike the offline setting, the online optimization problem deals with orders arriving sequentially over time, so that $\mathcal{O}$ is in general an unbounded stream; our experiments consider a finite horizon. At each time epoch t, a new order arrives that must be immediately assigned to a fulfillment center without knowledge of future orders. Each order o has an arrival time and a lifetime during which it must be processed; an order that has arrived and is alive is called active. The controller must make decisions based only on the current system state, balancing immediate cost optimization with preserving capacity for future orders. Suboptimal early assignments may lead to expensive solutions later in the time horizon. For example, if a controller assigns an order to the closest fulfillment center at 9 AM, customers placing orders at 2 PM might receive longer delivery time promises because their orders must now be fulfilled from more distant facilities. The 9 AM order could have been served from a more distant fulfillment center while still meeting the same delivery promise, preserving capacity at the closer facility.

3 Related Work

The order assignment problem that we study in this work generalizes GAP, as demonstrated through our reduction in Section 2.2. While the specific business context involves additional complexities unique to Amazon's logistics network – such as delivery station constraints, link capacities, and online decision-making requirements – GAP represents a fundamental special case of our problem.

Ross and Soland [18] introduce GAP as an extension of the classical assignment problem where multiple task assignments are constrained by agent resource limitations. They develop a branch-and-bound algorithm that solves GAP through a series of binary knapsack subproblems. It has been shown that GAP is NP-hard [15]. In general, the literature on the GAP is extensive. Martello and Toth [15] provide a general introduction to offline GAP. An early survey by Cattrysse and Wassenhove [7] outlines the foundational work in this area. More recently, Öncan [17] presented a comprehensive survey covering various solution approaches, including metaheuristics and variable neighborhood search methods. The latest developments are captured in Maniezzo et al. [14], which reviews state-of-the-art research and demonstrates the application of various mathematical programming techniques to GAP.

Shmoys and Tardos [22] gave a linear programming (LP) rounding algorithm for the minimum-cost version, while for the maximization version linear programming relaxations achieve a $(1-1/e)$-approximation [10]. Savelsbergh [19] presents a branch-and-price algorithm for GAP that combines column generation with branch-and-bound to solve a set partitioning formulation of the problem. Cohen et al. [8] present a simple approximation algorithm for GAP that translates any knapsack algorithm into a GAP approximation with ratio $(1 + \alpha)$, where α is the knapsack algorithm's approximation ratio. Yagiura [24] propose a very large-scale neighborhood search for the multi-resource GAP. Their approach demon-

strates how sophisticated neighborhood structures can effectively handle complex assignment variants. Mitrović-Minić and Punnen [16] proposed a variable neighborhood search algorithm for solving the multi-resource GAP. Similar to our approach, their method incorporates MIP solvers within a local search framework, though their multi-resource formulation addresses a different problem context.

Online General Assignment Problem. Alaei et al. [3] present a $(1 - 1/\sqrt{k})$-competitive algorithm for online stochastic GAP, where items arrive adversarially but distribution information is known in advance. Their work addresses the challenge of making assignment decisions when item sizes are revealed only after placement, providing theoretical guarantees for online assignment problems with stochastic elements. Albers et al. [4] develop improved competitive algorithms for online knapsack and GAP in the random order model, achieving ratios of 1/6.65 and 1/6.99 respectively through a sequential approach that employs specialized algorithms for different item classes. Liu et al. [13] study online GAP in the random order model, developing competitive algorithms that leverage historical information to improve assignment decisions. Xu et al. [23] study the online multi-dimensional GAP with predictions, where decision parameters like service costs and resource consumption can be predicted with bounded error. Closer to our application, real-time and LP-based order-fulfillment policies for e-commerce have been studied by Acimovic and Graves [1] and Jasin and Sinha [11]; our surcharge-based controller is moreover related to primal-dual and dual-price methods for online allocation [6,2].

4 Offline Algorithms

4.1 MIP Algorithm

The offline MIP algorithm employs a MIP solver on the formulation from Section 2.1. In principle this yields a minimum-cost assignment; in practice we run the solver with a time limit and a 0.5% relative MIP gap, so the returned incumbent may be suboptimal. While the problem is NP-hard, as it contains GAP as a special case (Section 2.2), this serves as our reference when complete information about all orders is available a priori; it returns the best solution found within the configured time limit (300 s or 1800 s in our experiments), which is within 0.5% of optimal if the solver terminates before the limit. The drawback of the algorithm is its computational cost, which grows quickly with instance size. Moreover, we will use this solver to solve various subproblems that arise in local search as well as in the online setting. In our implementation, we utilize FICO Xpress 9.5.4, a commercial MIP solver, as the underlying optimization engine for solving both the LP relaxation and MIP problems, configured with time limits, a relative MIP gap tolerance of 0.5%, single-threaded solves, and warm start capabilities.

Algorithm 1 Best Fit Algorithm. Priority ρ_{of} is the per-unit cost γ_{of} or the ratio γ_{of}/w_o.

procedure BestFit($\mathcal{O}, \mathcal{F}, \mathcal{D}, \mathbf{c}$)
  $\mathcal{P} \leftarrow \emptyset$ ▷ Initialize empty pairs list
  for $o \in \mathcal{O}$ **do**
    for $f \in \mathcal{F}$ **do**
      if IsReachable(o, f) **then**
        $\rho_{of} \leftarrow$ ComputePriority(o, f) ▷ γ_{of} or γ_{of}/w_o
        $\mathcal{P} \leftarrow \mathcal{P} \cup \{(o, f, \rho_{of})\}$
  Sort $\mathcal{P}$ by ρ_{of} in ascending order
  for $(o, f, \rho) \in \mathcal{P}$ **do**
    if o not already assigned **and** CapacitiesSatisfied($o, f, \mathbf{c}$) **then**
      set assignment of o to f in $\mathbf{X}$
  assign remaining orders in $\mathbf{X}$ to virtual fulfillment center
  return X

4.2 Heuristic Algorithms

Given the complexity and computational demands of the MIP solver, we explore several heuristic algorithms that trade optimality for speed and scalability. These heuristics aim to find good quality solutions quickly by making intelligent local decisions, making them particularly valuable for larger problem instances where the MIP solver becomes impractical. We present two main heuristic approaches: best fit, which makes assignments based on absolute costs or scaled-cost by consumption, and a batched approach which applies the MIP on subsets of orders iteratively. Additionally, we explore local search algorithms that iteratively improve existing solutions by exploring neighboring assignments.

Cost-based Best Fit The cost-based best fit algorithm implements a greedy heuristic that globally sorts all order-FC pairs by cost rather than processing orders sequentially. It first creates a list of all possible order-FC pairs with their associated costs, considering only reachable fulfillment centers. These pairs are then sorted by the per-unit cost γ_{of} (equivalently, since every capacity is measured in units of consumption, the cost per unit of capacity consumed), and the algorithm attempts to assign each order to its lowest-cost compatible FC while respecting capacity constraints. For each potential assignment, the algorithm checks if the FC, delivery station, and link capacities can accommodate consumption rate of the order (recall that every order has a fixed delivery station and hence if assigned to an FC also a fixed link that needs to be used). If no feasible assignment is found among the real FCs, the order is assigned to a virtual FC with a penalty cost. Pseudocode for the algorithm is shown in Algorithm 1. It is straightforward to see that the algorithm generates a feasible solution and that the complexity of the algorithm is $O(|\mathcal{O}||\mathcal{F}| \log(|\mathcal{O}||\mathcal{F}|))$, assuming $O(1)$ capacity checks.

Algorithm 2 Batch Mixed Integer Programming Algorithm

procedure BATCHMIPSOLVER($\mathcal{O}, \mathcal{F}, \mathcal{D}$, $\mathbf{c}$, χ)
  $\mathbf{c}_r \leftarrow \mathbf{c}$ ▷ Initialize remaining capacities for FC/DS/Links
  Sort $\mathcal{O}$ by consumption rate in descending order
  for $k = 0$ to $\lceil |\mathcal{O}|/\chi \rceil - 1$ **do**
    $\mathcal{O}_k \leftarrow \{o_{k\chi}, \ldots, o_{\min((k+1)\chi-1, |\mathcal{O}|-1)}\}$ ▷ Current batch
    // Construct subproblem P_k with remaining capacities $\mathbf{c}_r$
    $P_k \leftarrow \text{ConstructMIP}(\mathcal{O}_k, \mathcal{F}, \mathcal{D}, \mathbf{c}_r)$
    $\mathbf{X}_k \leftarrow \text{SolveMIP}(P_k, t_{\max})$ ▷ Time limit $t_{\max}$
    if $\mathbf{X}_k = \emptyset$ **then** ▷ no feasible incumbent within $t_{\max}$
      $\mathbf{X}_k \leftarrow \text{BestFitPerConsumption}(\mathcal{O}_k, \mathcal{F}, \mathcal{D}, \mathbf{c}_r)$
    set assignments of $\mathcal{O}_k$ in $\mathbf{X}$ using $\mathbf{X}_k$
    $\mathbf{c}_r \leftarrow \text{UpdateCapacities}(\mathbf{c}_r, \mathbf{X}_k)$
  return X

Consumption-scaled Cost-based Best Fit The consumption-scaled cost-based best fit algorithm operates similarly to the cost-based best fit algorithm, but with one key modification in how it prioritizes assignments. Instead of sorting order-FC pairs by the per-unit cost, it sorts them by γ_{of}/w_o; because dividing by the order size w_o favors larger orders, this ordering tends to place high-consumption orders first. The algorithm maintains the same process of checking FC, delivery station, and link capacity constraints before making assignments, and falls back to virtual FC assignments when no feasible real FC assignment exists. Like its predecessor, the algorithm generates feasible solutions and has a complexity of $O(|\mathcal{O}||\mathcal{F}| \log(|\mathcal{O}||\mathcal{F}|))$.

Consumption-based Batching The batch MIP algorithm addresses the scalability limitations of the MIP solver by breaking down the problem into smaller, more manageable batches. The algorithm processes orders sequentially in fixed-size batches, maintaining a cumulative solution while respecting capacity constraints across batches. The process begins by sorting the orders by consumption and creating batches of size χ, where χ is a tuning parameter. The algorithm sorts by consumption because experiments from Section 6 indicate an advantage to processing high-consumption orders first. A batch is processed by creating a subproblem that considers the current state of the resources of the system. For each batch, we construct a smaller MIP using the remaining capacities after accounting for all previously assigned orders. The first batch begins with the original input capacities, while subsequent batches work with progressively reduced capacities based on prior assignments.

If the MIP solver fails to find a solution for a batch within a given time limit, it falls back to a heuristic approach (best fit per consumption). The batch size parameter χ provides a flexible way to balance solution quality and computational efficiency – larger batches potentially yield better solutions but require more computation time, while smaller batches offer faster processing but may

Algorithm 3 Extended Swap Local Search Algorithm

procedure ExtendedSwap($\mathcal{O}, \mathcal{F}, \mathcal{D}, \mathbf{c}, \mathbf{X}_0, \kappa, T$)
  $\mathbf{X}^* \leftarrow \mathbf{X}_0$ ▷ Best solution found
  for $t = 1$ to T **do** ▷ Iterate T times
    $\mathcal{O}_t \leftarrow$ SampleOrders($\mathcal{O}, \kappa$) ▷ Random κ orders
    $\mathbf{X}_{\text{new}} \leftarrow \mathbf{X}^*$ ▷ Current solution
    // remove selected orders and compute remaining capacities
    $\mathbf{c}_r \leftarrow$ ComputeRemainingCapacities($\mathbf{X}^*, \mathbf{c}, \mathcal{O} \backslash \mathcal{O}_t$)
    $P_t \leftarrow$ ConstructMIP($\mathcal{O}_t, \mathcal{F}, \mathcal{D}, \mathbf{c}_r$) ▷ Construct and solve subproblem
    $\mathbf{X}' \leftarrow$ SolveMIP($P_t, t_{\max}$)
    update assignments of $\mathcal{O}_t$ in $\mathbf{X}_{\text{new}}$ using $\mathbf{X}'$
    if Cost($\mathbf{X}_{\text{new}}$) < Cost($\mathbf{X}^*$) **then** ▷ Update solution if improvement found
      $\mathbf{X}^* \leftarrow \mathbf{X}_{\text{new}}$
  return $\mathbf{X}^*$

lead to more locally optimal decisions. Pseudocode for the algorithm can be found in Algorithm 2.

4.3 Local Search Algorithms

Our local search algorithm is an extended swap algorithm in the spirit of large neighborhood search [21] and the ruin-and-recreate principle [20]. In each iteration, the algorithm randomly selects κ orders (where κ is a tuning parameter) and attempts to find better assignments for them collectively. The process begins by removing these selected orders from their current assignments and thus freeing up capacity at the resources that they used. Afterwards the remaining capacities for all fulfillment centers, delivery stations, and links in the network are recalculated. A smaller MIP problem is then constructed and solved under a time limit to reassign just these κ orders while respecting the updated/remaining capacity constraints. If the new assignments result in a lower total cost, the global solution is updated by incorporating these new assignments and recalculating all resource utilizations. The algorithm continues this process for a pre-specified number of iterations. The algorithm provides a very general local search neighborhood – when $\kappa = |\mathcal{O}|$, it reduces to solving the complete MIP problem, while smaller values of κ yield more restricted neighborhoods. Pseudocode can be found in Algorithm 3.

5 Online Algorithms

The online algorithms process orders sequentially as they arrive, making immediate assignment decisions without knowledge of future orders. We implement three variants: a basic greedy algorithm, a capacity-aware surcharge approach (Lagrange), and a reoptimization method. All variants can be enhanced with periodic batch reoptimization, where previously assigned orders within a time window are reconsidered and potentially reassigned to improve solution quality.

Algorithm 4 Greedy Online Controller

procedure GreedyOnlineController($\mathcal{O}, \mathcal{F}, \mathcal{D}, \mathbf{c}$)
  $\mathbf{c}_r \leftarrow \mathbf{c}$ ▷ Initialize remaining capacities for FC/DS/Links
  for $o \in \mathcal{O}$ in arrival order **do** ▷ Process orders sequentially by arrival time
    $\tilde{\mathcal{F}}_o \leftarrow \emptyset$ ▷ Initialize feasible facilities
    for $f \in \mathcal{F}_o$ **do**
      if sufficient remaining capacity in FC f, Link (f, d_o) and DS d_o **then**
        $\tilde{\mathcal{F}}_o \leftarrow \tilde{\mathcal{F}}_o \cup \{f\}$
    if $\tilde{\mathcal{F}}_o \neq \emptyset$ **then**
      $f^* \leftarrow \arg\min_{f \in \tilde{\mathcal{F}}_o} \gamma_{of} w_o$ ▷ Select lowest cost fulfillment center
      assign order o to fulfillment center f^*
      update $\mathbf{c}_r$ ▷ Decrease remaining capacities
    else
      assign order o to virtual fulfillment center
  return X

5.1 Greedy Online Controller

The greedy online controller (Algorithm 4) processes orders sequentially as they arrive. For each order, it first identifies all reachable fulfillment centers whose remaining capacities (FC capacity, DS capacity, and link capacity) are sufficient to accommodate the consumption rate of the current order. These feasible facilities are then ranked by their total assignment cost (fulfillment center cost multiplied by consumption rate), and the lowest-cost option is selected. If no feasible options exist, the order must be assigned to the virtual fulfillment center, incurring the corresponding penalty cost. While computationally efficient, this greedy approach suffers from myopic decision-making: by always selecting the lowest-cost option, it may prematurely exhaust capacity at strategically located facilities, forcing later orders to use more expensive facilities or the virtual fulfillment center.

5.2 Capacity-Aware Surcharges

This controller (labeled Lagrange in Table 1 and Figure 4) augments the greedy approach with per-unit surcharges that anticipate future resource scarcity. Inspired by shadow-price ideas from online optimization [12], it maintains a surcharge λ_r for each resource type (fulfillment centers, delivery stations, and links) that is added to the per-unit assignment cost. We stress that λ_r is a heuristic penalty price, not a dual multiplier of the offline program. To estimate future resource requirements, the controller employs a rolling horizon approach. It maintains a history of consumption patterns within a time window (set to one-tenth of the total simulation duration) and extrapolates recent usage to project demand over the remaining horizon. When these projections indicate potential capacity shortages, the corresponding surcharges are increased (a one-sided update) to discourage further assignments to these resources:

$$\lambda_r \leftarrow \lambda_r + \alpha \max(0, \mathbf{c}_r^{\text{projected}} - \mathbf{c}_r^{\text{remaining}})$$

where λ_r is the surcharge for resource r, α is a step size (typically 0.1), $\mathbf{c}_r^{\text{remaining}}$ is the currently remaining capacity and $\mathbf{c}_r^{\text{projected}}$ is the projected consumption at the end of the simulation based on the assignments that have been done in the current time window. When assigning an order o, it is placed at the reachable fulfillment center minimizing the surcharge-augmented per-unit score:

$$\gamma_{\text{effective}}(f) = \gamma_{of} + \lambda_f + \lambda_{d_o} + \lambda_{(f,d_o)}$$

where λ_f, λ_{d_o}, and $\lambda_{(f,d_o)}$ are the surcharges for the fulfillment center, delivery station, and link, respectively. The booked cost of the assignment is $\gamma_{of}\, w_o$; since every capacity is measured in units of w_o, minimizing this per-unit score is equivalent to minimizing $w_o(\gamma_{of} + \lambda_f + \lambda_{d_o} + \lambda_{(f,d_o)})$. This makes heavily-utilized resources appear progressively more expensive, encouraging a balanced distribution of orders and helping prevent premature capacity exhaustion.

5.3 Batch Re-Assignment

The re-assignment strategy periodically re-optimizes previous assignments. The controller operates with a reoptimization interval Δt: during normal operation, it makes greedy online assignments for new orders. At periodic timestamps $t = k\Delta t$, it triggers a reoptimization phase for orders assigned within the recent time window. The reoptimization considers the remaining capacities after accounting for fixed assignments outside the time window:

$$\mathbf{c}_r^{\text{available}} = \mathbf{c}_r^{\text{total}} - \sum_{o \in \tilde{\mathcal{O}}_t} w_{or} x_{or}$$

where $x_{or} \in \{0, 1\}$ indicates whether order o uses resource r, and $\tilde{\mathcal{O}}_t$ is the set of previous orders with arrival time $\leq t$ that are not alive anymore. Similarly, capacities for links and delivery stations are computed. The reoptimization window size and frequency can be tuned to balance solution quality against computational cost. After solving this optimization, we update the assignments of all orders in the batch and resume the online controller. Only orders that are still active, i.e. within the current window and not yet physically fulfilled, are reconsidered; already-processed orders are fixed, so the reassignment is operationally admissible. Each reoptimization is bounded by the subproblem time limit, which keeps per-decision latency within the operational cadence, and the greedy and LAGRANGE controllers correspond to the special case without reoptimization.

For solving the batch reoptimization subproblem, any of the previously discussed offline optimization approaches can be employed: best-fit algorithms for fast solutions, local search for iterative improvements, or the batch MIP solver for higher solution quality. Additionally, all algorithms can be extended with trajectory constraints: for each resource r and time t, a hard constraint requires that cumulative consumption not exceed a target utilization $\overline{u}_r(t)$ derived from historical patterns and forecasts, and determining these target bands (trajectory generation) is itself a separate optimization problem.

Table 1. Summary of algorithms. LS = local search (Alg. 3, 20 iterations, neighborhood size $\kappa = 200$ unless noted), RO = periodic reoptimization. Numeric MIP suffix = time limit in seconds, Sχ = batch size χ. MIP solves use the stated time limit and a 0.5% gap, so they are not guaranteed optimal.

Name	Type	Description
Offline Algorithms (Section 4)		
MIP300	MIP	MIP solver, 300 s time limit
MIP1800	MIP	MIP solver, 1800 s time limit
BestFitCost	Heuristic	Best fit sorted by cost
BestFitConsumptionScaled	Heuristic	Best fit sorted by cost/consumption
BatchOfflineS200	Heuristic	Batch MIP, batch size 200
BatchOfflineS200+LS	Hybrid	BatchOfflineS200 + local search
MIP300+LS	Hybrid	MIP300 + local search
Online Algorithms (Section 5)		
OnlineGreedy	Online	Greedy controller
Lagrange	Online	Capacity-aware surcharge controller
ROMIP	Online+RO	Reoptimization using full MIP
ROBatchOffline	Online+RO	Reoptimization w/ batch MIP (batch size 20)
ROBestFitCS	Online+RO	Reoptimization w/ cons.-scaled best fit

6 Experimental Evaluation

Table 1 provides an overview of all algorithms evaluated in this section, mapping each name (as used in figures) to its definition.

Methodology. We performed our implementations using Python 3. All of our experiments were run on a machine equipped with two Intel(R) Xeon(R) Platinum 8259CL CPU with 24 cores each running at 2.50GHz with 71.5 MB L3 Cache and 373 GB of main memory. By default, we execute up to 48 experiments in parallel. To compare algorithms, we use *performance profiles* [9]. The y-axis shows the fraction of instances where an algorithm's objective is within a factor τ of the best: $\#\{\text{objective} \leq \tau \cdot \text{best}\}/\#\text{instances}$. At $\tau = 1$, we read off how often an algorithm achieves the best result. Unless otherwise mentioned the MIP solver type time limit for any type of *subproblem* that we solve is set to ten seconds. Note that the MIP solver can finish considerably earlier. In general, unless otherwise mentioned we use twenty iterations of local search per (sub)problem. Increasing the number of iterations typically yields better quality solutions but comes at the cost of longer computational running times.

Instance Generation. We generate artificial instances that match real-world applications closely. While we would prefer to evaluate on actual Amazon logistics data, we do not have approval to publish results using proprietary operational data. Our synthetic instances are designed based on operational insights from Amazon's logistics network, incorporating capacity distributions, order arrival patterns, and network topologies observed in practice. In particular, ex-

ponential interarrival times model the Poisson-like order streams observed in e-commerce, capacity-proportional order generation reflects the geographic demand distribution across delivery stations, and the load factor controls the tightness of capacity constraints to match realistic operational regimes. Concretely, the load factor is the ratio of total expected order consumption to total resource capacity, so a value above one yields an oversubscribed instance in which some orders must be served by the virtual fulfillment center.

The instance generation process begins by establishing the network structure. Our generator has several parameters: the number of facilities, number of delivery stations and number of orders, mean order lifetime (default 100u, where u is a simulation time unit), reoptimization interval (default 100u), simulation duration (default 600u), load factor (default 1.2), FC reachability probability (default 0.7), a lower bound for the link capacity l_{lb} (default 10) and a respective upper bound l_{ub} (default 100). The generator creates a network plan by generating the required number of facilities and delivery stations. Every fulfillment center is connected to every delivery station, i.e. the link topology is complete bipartite, and each link capacity is drawn independently from $U[l_{\text{lb}}, l_{\text{ub}}]$; sparsity enters only through the per-order reachability sets, since order o can use link (f, d_o) only if f is one of its reachable fulfillment centers. The capacity of a DS is then set to the sum of the incoming link capacities, while the capacity of an FC is set to the sum of the outgoing link capacities. With this construction the fulfillment-center and delivery-station capacity constraints are implied by the link-capacity constraints (summing a fulfillment center's outgoing-link constraints yields its capacity bound, and symmetrically for a delivery station). Each offline instance therefore separates into independent per-delivery-station GAP subproblems, so our evaluation spans a large, heterogeneous collection of GAP subproblems that is amenable to parallelization. Regimes in which node capacities bind require independently drawn node capacities, which we examine below as a robustness check.

The subsequent order generation process proceeds by creating orders for each delivery station in proportion to its capacity: the number of orders for DS d is $\lceil \frac{\mathbf{c}_d}{\sum_{d'} \mathbf{c}_{d'}} \cdot |\mathcal{O}| \rceil$. Order interarrival times follow an exponential distribution (Poisson process). Consumption rates and lifetimes are each drawn from exponential distributions with the respective mean parameters. Each order draws a single consumption rate that we use for all resources it occupies (i.e. $w_{of} = w_{oe} = w_{od}$), with mean set so that the total expected demand equals the load factor times the total capacity. For each order, the set of reachable FCs is determined by including each FC independently with the reachability probability; orders without any reachable FC receive a connection to a random FC. The cost γ_{of} for each order-FC pair is $10+90r$, where $r \sim \text{Uniform}[0, 1]$ is drawn independently for each pair. The virtual fulfillment center incurs a per-unit penalty $M = 10^4$. For our experiments, we generate three instances for each combination of $n_f \in \{4, \ldots, 20\}$ and $n_d \in \{4, \ldots, 20\}$ (all integer values, i.e. $17 \times 17 \times 3 = 867$ offline instances). Each instance contains 500 orders for offline experiments (enabling MIP solver comparisons) and 10,000 orders for online algorithm evaluation. Due to proprietary

constraints, we cannot publicly release the source code or instance generator. However, all algorithms are given as pseudocode with complete parameter specifications, and the instance generation procedure is fully specified (distributions, defaults, construction logic), enabling independent reproduction.

As a robustness check, we additionally ran a second capacity regime in which fulfillment-center and delivery-station capacities are drawn independently as β times the incident link-capacity sums, with $\beta \sim U[0.5, 0.8]$ per node, so that the node constraints bind rather than being implied by the link constraints; instances are paired across regimes at the same demand-to-capacity ratio. In the coupled regime the binding resource shifts from the links to the nodes: calling a resource *saturated* when its assigned load reaches at least 99% of its capacity at the offline MIP solution, with link fractions taken over all $|\mathcal{F}| \times |\mathcal{D}|$ pairs of the complete bipartite topology, mean link saturation falls from 52% to 4.7%, while fulfillment-center and delivery-station saturation rise from 31% to 67% and from 45% to 56%, respectively. On these paired instances, the algorithm ranking and all pairwise comparisons are preserved; the only effect is that the gaps between heuristics compress, since tighter node capacities push more demand to the virtual fulfillment center for every method. Hence, we focus on presenting the results for the regime described above, in which fulfillment-center and delivery-station capacities equal the sums of their incident link capacities.

Running Time. Non-MIP algorithms (best fit, greedy) run in under one second on our instances. MIP-based methods are dominated by their configured time limits: 10 s per subproblem for batch and local search iterations, 300 s or 1800 s for the full offline MIP. For example, BatchOfflineS200 uses at most 30 s of MIP time (three batches), while MIP300+LS uses at most 500 s (300 s MIP + 20×10 s LS). These MIP budgets serve as a consistent measure of solver effort across methods.

6.1 Evaluation

Our analysis focuses on three key aspects: the performance of offline algorithms, the effectiveness of local search improvements, and the behavior of online controllers. For each category, we examine solution quality using performance profiles. For offline algorithms, we compare the MIP solver against our heuristic approaches, analyzing how solution quality trades off against running time as problem size increases. The local search evaluation focuses on how different parameter settings affect solution improvement. For online controllers, we examine how different strategies maintain good solution quality.

Offline Algorithms. Our experimental results show that the batch offline approach BatchOfflineS200 (i.e. batch size 200) achieves comparable solution quality to the full MIP solver MIP300 running for 300 seconds, while offering substantial advantages in computational efficiency and control. The batch algorithm requires at most 30 seconds of MIP solver time (by configuration there are

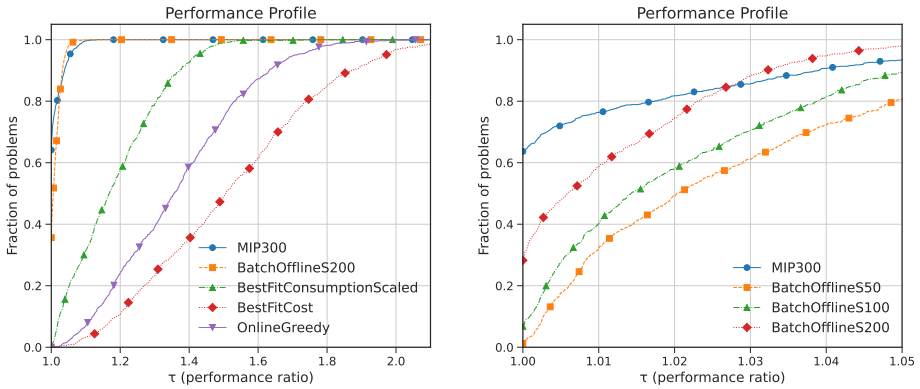


Fig. 2. Left: Performance profiles of offline algorithms (higher y-value = better); see Table 1 for algorithm names. Right: Impact of batch sizes (50–200) on solution quality. Note the two panels use different τ ranges.

three batches), making it substantially more practical. The performance profiles in Figure 2 demonstrate this parity in solution quality. Notably, ordering by γ_{of}/w_o (BestFitConsumptionScaled), which prioritizes larger orders, outperforms per-unit-cost ordering (BestFitCost). As the instances are oversubscribed (load factor 1.2) and the virtual-fulfillment-center penalty dominates the objective, this gap primarily reflects how much demand each heuristic keeps in-house rather than the cost of the placements themselves. While we included the online greedy algorithm OnlineGreedy run as an offline algorithm for comparison, it performs markedly worse than the consumption-scaled best fit approach. Moreover, Figure 2 considers the impact of batch size on solution quality. As expected larger batch sizes increase solution quality.

Observation 1. In the oversubscribed regime, prioritizing larger orders (BestFitConsumptionScaled) keeps more demand in-house than per-unit-cost ordering (BestFitCost); since outsourcing to the virtual fulfillment center dominates the objective, this lowers total cost.

Observation 2. Batch algorithms achieve comparable solution quality to 300-second MIP runs at a fraction of the MIP time budget.

Local Search. Figure 3 shows performance profiles comparing different offline algorithms when combined with local search. The local search algorithm demonstrates consistent ability to improve solutions across different input algorithms. When applied to faster but lower-quality heuristics like best-fit consumption BestFitConsumptionScaled, the improvements are particularly substantial. For solutions from more sophisticated algorithms, the improvements

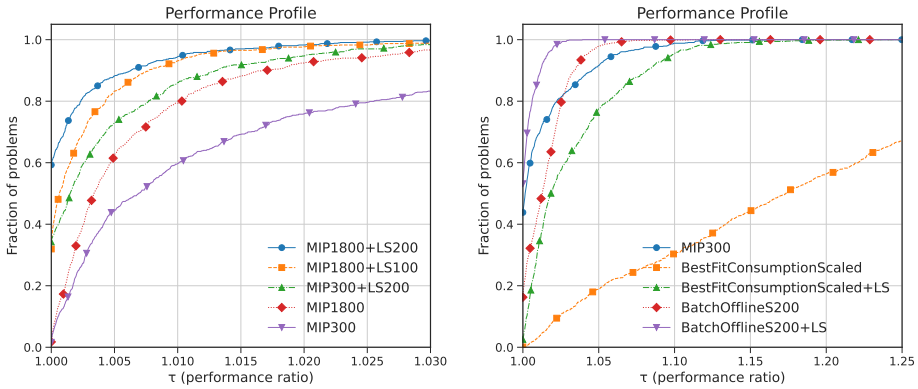


Fig. 3. Impact of local search (LS) on solution quality. Left: MIP solutions with and without LS for different MIP time limits (300 s, 1800 s). Right: LS improvements on best-fit and batch algorithm solutions. See Table 1 for algorithm names.

remain substantial though more modest. A notable example is the batch offline scenario, where applying local search for 20 iterations with neighborhood size 200, i.e. BatchOfflineS200+LS to solutions from the batch algorithm BatchOfflineS200 with batch size 200 (requiring three 10-second MIP solver runs) achieves better results than running the MIP solver alone for 300 seconds. This highlights how local search can effectively complement other optimization approaches, offering a favorable trade-off between solution quality and computational overhead. The local search algorithm also demonstrates effectiveness in improving solution quality for solutions obtained from long-running MIP solvers. Our experiments show that running local search for twenty iterations with neighborhood sizes κ of 100 or 200 yields consistent improvements. As shown in Figure 3, the algorithm can enhance solutions even when starting from a high-quality initial solution computed by MIP1800. Most notably, running the offline MIP for 300 seconds followed by local search outperforms MIP1800. Note that this requires substantially less total computation time: 300 seconds for initial MIP plus 20 iterations $\times$ 10 seconds per iteration $=$ 500 seconds total, compared to 1800 seconds for the long-running MIP. However, it is worth noting that the relative improvements in this setting are modest, as initial solutions from the offline MIP algorithm achieve high quality.

Observation 3. Local search provides a scalable solution quality vs. runtime trade-off, improving solutions across all algorithms.

Online Algorithms. We now examine the performance of various online algorithms (Figure 4) in a scenario where all online algorithms utilize a linear trajectory for resource protection, i.e. a target utilization curve that limits how

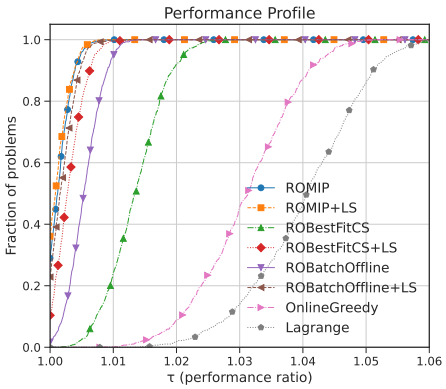


Fig. 4. Performance profiles of online controllers. RO = periodic reoptimization; +LS = with additional local search. See Table 1 for algorithm names.

much of each resource's capacity may be consumed by time t (see Section 5.3). The linear trajectory we use here starts with 1% of capacity and ends at 100% of capacity. This trajectory is a fixed protocol choice applied uniformly to all controllers rather than a focus of this study; a systematic evaluation of alternative trajectories is left to future work. Moreover, we use 10,000 orders and configure the batching systems so that the average amount of orders in an active set of orders is 50: by Little's law the expected number of concurrently active orders equals the arrival rate times the mean order lifetime, and we use a mean order lifetime of 3 u to obtain this value. Among all tested approaches, the batch approach ROMIP consistently delivers the best performance. Improvements obtained by applying additional local search (20 iterations, 1s time limit per MIP, neighborhood size 20) on batches of active orders of this algorithm are minimal since the small subproblems are already solved mostly to optimality. ROBatchOffline with an internal batch size of 20, time limit of 10s per MIP, shows strong performance, particularly when enhanced with local search, bringing its results very close to ROMIP's quality. ROBestFitCS demonstrates consistently better results compared to OnlineGreedy and Lagrange approaches, which perform poorest in this context. Notably, while local search provides limited benefits to ROMIP, it substantially improves the solutions of both ROBatchOffline and ROBestFitCS algorithms. Lastly, note that even modest improvements in solution quality yield substantial savings in operational costs in large-scale logistics operations.

Observation 4. Periodic reoptimization consistently outperforms pure online approaches, and local search further improves lower-quality initial solutions.

7 Conclusion

We investigated hybrid optimization approaches for Amazon's order assignment problem in offline and online settings. Our experiments show that (1) consumption-scaled approaches outperform cost-based methods in oversubscribed settings, (2) local search provides scalable improvements, (3) batch MIP processing matches a 300-second MIP at a fraction of the cost, and (4) periodic reoptimization consistently outperforms traditional online methods. These findings also hold in a capacity regime where fulfillment-center and delivery-station capacities bind rather than being implied by the link capacities, indicating they are not an artifact of the instance construction. Future work includes parallelization strategies for local search, replacing hard trajectory constraints with penalty-based formulations, and theoretical analysis of competitive ratios and guarantees. On the empirical side, we plan to ablate the trajectory constraint, scale to substantially larger instances, evaluate underloaded instances in which assignment costs rather than outsourcing penalties drive the objective, and compare against other MIP solvers.

Acknowledgments. The text of this paper was polished with the help of the large language model Claude Opus 5.5 [5]; all technical content, algorithms and experiments are the work of the authors.

Disclosure of Interests. The authors are employees of Amazon. The authors have no other competing interests to declare that are relevant to the content of this article.

References

1. Acimovic, J., Graves, S.C.: Making better fulfillment decisions on the fly in an online retail environment. Manufacturing & Service Operations Management **17**(1), 34–51 (2015). https://doi.org/10.1287/msom.2014.0505
2. Agrawal, S., Wang, Z., Ye, Y.: A dynamic near-optimal algorithm for online linear programming. Operations Research **62**(4), 876–890 (2014). https://doi.org/10.1287/opre.2014.1289
3. Alaei, S., Hajiaghayi, M., Liaghat, V.: The online stochastic generalized assignment problem. In: Approximation, Randomization, and Combinatorial Optimization (RANDOM). LNCS, vol. 8096, pp. 11–25. Springer (2013). https://doi.org/10.1007/978-3-642-40328-6_2
4. Albers, S., Khan, A., Ladewig, L.: Improved online algorithms for knapsack and GAP in the random order model. Algorithmica **83**(6), 1750–1785 (2021). https://doi.org/10.1007/S00453-021-00801-2
5. Anthropic: Claude Opus 5.5. Large language model, https://www.anthropic.com/claude (2026), accessed: 2026-11-16
6. Buchbinder, N., Naor, J.: The design of competitive online algorithms via a primal-dual approach. Foundations and Trends in Theoretical Computer Science **3**(2–3), 93–263 (2009). https://doi.org/10.1561/0400000024
7. Cattrysse, D.G., Van Wassenhove, L.N.: A survey of algorithms for the generalized assignment problem. European Journal of Operational Research **60**(3), 260–272 (1992). https://doi.org/10.1016/0377-2217(92)90077-M

8. Cohen, R., Katzir, L., Raz, D.: An efficient approximation for the generalized assignment problem. Information Processing Letters **100**(4), 162–166 (2006). https://doi.org/10.1016/J.IPL.2006.06.003
9. Dolan, E.D., Moré, J.J.: Benchmarking Optimization Software With Performance Profiles. Mathematical Programming **91**(2), 201–213 (2002). https://doi.org/10.1007/S101070100263
10. Fleischer, L., Goemans, M.X., Mirrokni, V.S., Sviridenko, M.: Tight approximation algorithms for maximum general assignment problems. In: ACM-SIAM Symposium on Discrete Algorithms (SODA). pp. 611–620. SIAM (2006). https://doi.org/10.1145/1109557.1109624
11. Jasin, S., Sinha, A.: An LP-based correlated rounding scheme for multi-item ecommerce order fulfillment. Operations Research **63**(6), 1336–1351 (2015). https://doi.org/10.1287/opre.2015.1441
12. Liakopoulos, N., Destounis, A., Paschos, G., Spyropoulos, T., Mertikopoulos, P.: Cautious regret minimization: Online optimization with long-term budget constraints. In: International Conference on Machine Learning. vol. 97, pp. 3944–3952. PMLR (2019)
13. Liu, H., Zhang, H., Luo, K., Xu, Y., Xu, Y., Tong, W.: Online generalized assignment problem with historical information. Computers & Operations Research **149**, 106047 (2023). https://doi.org/10.1016/J.COR.2022.106047
14. Maniezzo, V., Boschetti, M.A., Stützle, T.: The Generalized Assignment Problem, pp. 3–33. Springer International Publishing, Cham (2021). https://doi.org/10.1007/978-3-030-70277-9_1
15. Martello, S., Toth, P.: Knapsack problems: algorithms and computer implementations. John Wiley & Sons, New York, NY (1990)
16. Mitrovic-Minic, S., Punnen, A.P.: Local search intensified: Very large-scale variable neighborhood search for the multi-resource generalized assignment problem. Discrete Optimization **6**(4), 370–377 (2009). https://doi.org/10.1016/J.DISOPT.2009.04.004
17. Öncan, T.: A survey of the generalized assignment problem and its applications. INFOR: Information Systems and Operational Research **45**(3), 123–141 (2007). https://doi.org/10.3138/INFOR.45.3.123
18. Ross, G.T., Soland, R.M.: A branch and bound algorithm for the generalized assignment problem. Mathematical Programming **8**(1), 91–103 (1975). https://doi.org/10.1007/BF01580430
19. Savelsbergh, M.W.P.: A branch-and-price algorithm for the generalized assignment problem. Operations Research **45**(6), 831–841 (1997). https://doi.org/10.1287/OPRE.45.6.831
20. Schrimpf, G., Schneider, J., Stamm-Wilbrandt, H., Dueck, G.: Record breaking optimization results using the ruin and recreate principle. Journal of Computational Physics **159**(2), 139–171 (2000). https://doi.org/10.1006/jcph.1999.6413
21. Shaw, P.: Using constraint programming and local search methods to solve vehicle routing problems. In: Principles and Practice of Constraint Programming (CP). Lecture Notes in Computer Science, vol. 1520, pp. 417–431. Springer (1998). https://doi.org/10.1007/3-540-49481-2_30
22. Shmoys, D.B., Tardos, Éva.: An approximation algorithm for the generalized assignment problem. Mathematical Programming **62**, 461–474 (1993). https://doi.org/10.1007/BF01585178
23. Xu, Y., Zheng, J., Chen, G., Zhu, X., Yao, Z.: Online multi-dimensional generalized assignment problem with predictions. Theoretical Computer Science **1056**, 115505 (2025). https://doi.org/10.1016/J.TCS.2025.115505

24. Yagiura, M., Iwasaki, S., Ibaraki, T., Glover, F.W.: A very large-scale neighborhood search algorithm for the multi-resource generalized assignment problem. Discrete Optimization **1**(1), 87–98 (2004). https://doi.org/10.1016/J.DISOPT.2004.03.005