Prioritizing Order Scheduling Under Resource Constraints in a CPG Supply Chain

Jonathan Ling
Department of Management Science Engineering, Stanford University, Stanford, CA 94305, jonling@stanford.edu

In supply chain distribution, customer orders are fulfillable only if all the product and logistical resources required to do so are available. Therefore, if a particular resource lacks the capacity to contribute to an order that requires it by the customer's desired delivery date, that order must be rescheduled. Three key resources required are the product, carrier (i.e., truck) and dock (i.e., warehouse loading space). To avoid wasting scheduling efforts, schedulers must decide whether to schedule or reschedule each order with respect to the customers' desired delivery date before taking that action. Further, in light of resource contention constraints, schedulers should ideally prioritize orders that are worth more to the business to be scheduled rather than rescheduled. This paper formulates a binary integer program as a variant of the knapsack problem in operations research, to optimize the prioritization sequence of orders while also prescribing whether to schedule or reschedule each one.

Key words: resource contention; supply chain distribution; scheduling; knapsack problem

1. Introduction

Supply chain operations, particularly in the high-volume Consumer Packaged Goods (CPG) industry, are characterized by the need for systematic efficiency at scale. At the intersection of manufacturing and distribution, customer orders are to be scheduled in light of the product and logistical resources needed for that order. It is common practice for customers to request a specific delivery date, with the CPG company's performance measured as the percentage of orders shipped on each order's requested date. This may be formalized as a service-level agreement (SLA) metric in the contract between the CPG company and the customer. Therefore, it is ideal to ship as many orders as possible on each order's requested date, or at least up to the SLA metric, in a cost-effective manner.

However, this is not always possible to do as the resources required by the orders may be limited. In addition to product resources, with distinct product types commonly known as SKUs (stock keeping units), key logistical resources are carriers and dock space. The carrier provides the truck that ships the product, while the dock space is a loading area at a warehouse where the product is transferred to the truck. A CPG company typically has pre-defined routes, commonly known as lanes, between their warehouses and customer locations, such as customer warehouses or stores.

Additionally, they will often outsource their truck fleet used for distribution, and have contracts with multiple third party trucking carriers for each lane it has, for which the truck rate (in dollars per mile) and number of trucks available per lane per day is specified. In this case, each carrier resource should be thought of as the sum of all trucks available across the different carriers for a specific lane.

For a set of orders that share a specific limited resource on a particular requested date, only some can be scheduled for that date, while the rest must be rescheduled. Deciding which of these two actions to take for each order should be based on the order's customer SLA, as well as the opportunity cost to the business of rescheduling rather than scheduling the order on the requested date. Accounting for product and logistical resource constraints, while optimizing for the customer's SLA and the CPG company's costs, will provide order-level scheduling directives that will ensure good customer service in terms of on-time delivery, and minimize associated costs to the CPG company, and avoid unnecessary rescheduling efforts from not being informed of resource contention constraints. For the scope of this paper, we will concentrate on optimizing for costs alone, and leave the SLA constraints as an extension for further research.

A systematic way to plan the scheduling of these orders is to use tools from operations research, and particularly in mathematical programming, to optimize the objectives of this problem in light of the problem's constraints. The nature of this problem is related to the knapsack problem in operations research. This paper will formulate a mathematical program based on the knapsack problem to solve the scheduling problem as described.

2. Literature Review

The knapsack problem is a well-studied problem in operations research with which the described scheduling problem has strong similarities. Numerous references on the knapsack problem can be found in Assi and Haraty (2018). In the simplest case, it involves choosing a subset of n items indexed by $j = 1, ..., n$, each with weight w_j and value v_j, to fit into a knapsack with a total weight constraint b, such that the total value sum of the selected items is maximized.

In the same way, each resource in the scheduling problem can be thought of as a knapsack $i = 1, ..., m$, with one knapsack per resource - that is, every distinct SKU, carrier (or lane, in the case of multiple carriers per lane) and dock space corresponds to its own knapsack. Analogously, for each knapsack i, its weight limit b_i is the available capacity of that resource; for example, if there are 10 units of product A available to be scheduled, then the 'weight limit' of product A's corresponding knapsack would be 10 units.

Continuing this analogy, the items j correspond to the orders, the items' weights w_j are those orders' resource requirements (e.g., 5 units of product A, 1 unit of carrier B and 1 unit of dock

space C), and each item's value v_j is the opportunity cost of not scheduling that order on the customer's desired delivery date. If this item can fit into the three types of knapsacks (product, carrier and dock) from which that order needs resources, then it can be scheduled. Otherwise, it must be rescheduled, and the business would pay the opportunity cost of doing so.

The knapsack problem then becomes trying to maximize the value to the business of all unscheduled orders that can be scheduled by the customer's delivery date, given the constraints on all knapsacks.

Further, for a time horizon with multiple dates, there would be a knapsack of each type for each date. Namely, each SKU-warehouse-date combination, carrier-date combination and dock-appointment slot combination would have their own corresponding knapsack. For CPG companies that employ multiple carriers on the same lane, carrier-date constraints would be replaced by lane-date constraints.

Though similar to the canonical single-knapsack problem, this scheduling problem differs in several ways:

1. It has multiple constrained resources, which needs to be modeled either as multiple knapsacks or multiple constraints, but the canonical knapsack problem only involves one knapsack, and one type of constraint (e.g., weight).
2. The knapsack problem is effectively being solved multiple times - once for each resource that each order requires. The algorithm then has to check that all the resources that an order needs are assigned to it to allow it to be scheduled on time. The formulation should ensure that the objective function correctly incentivizes the coordination of assigning all of an order's required resources, and not treat those multiple knapsacks as unrelated.
3. For product resources, there is an added complication that taking products from a specific date will impact the available capacity of that product at all earlier dates.

The literature presents a number of generalizations and variants of the knapsack problem, though the specific needs of this paper's scheduling problem require further configuration. The closest variants are described below.

The *multiple knapsack problem* seeks to assign a subset of n items to k distinct knapsacks, with the same type of objective function and constraints as the canonical knapsack problem. Bounds have been well studied, such as in Chekuri and Khanna (2005). However, it addresses point 1 above but not point 2 or 3.

The *nested knapsack problem* subdivides knapsacks into inner knapsacks within each one, allowing nested constraints as similar to before. Bounds have also been well studied, such as in Johnston and Khan (1995). It partially addresses point 2 by supposing that we can have three layers of knapsacks (product, carrier and dock). However, it does not fully address point 2, because there

is no natural ordering of those three resource categories where one completely fits into the second which in turn also completely fits into the third.

The *multiply-constrained knapsack problem*, also known as the *multi-dimensional knapsack problem*, involves a single knapsack but multiple types of constraints, such as both a weight and volume limit. Bounds have been studied in Fréville and Hanafi (2005). In the scheduling problem, this would result in one constraint for each SKU-plant-date, lane-date and dock-appointment slot combination. This would address points 1 and 2, however it does not satisfy point 3.

However, it is still relatively tractable to formulate and solve the problem at hand. The knapsack problem and its variants are well-studied, and there are thus multiple common ways of solving it. Three common methods are linear programming, reinforcement learning and dynamic programming. Linear programming is the most common approach to solving knapsack-like problems, and its formulation will be used in this paper, as it is a deterministic method that is also relatively simple and flexible to formulate and understand. Reinforcement learning is a method that experimentally searches the state space of possible solutions in search of the optimal point. Its formulation comprises an action space, state space and reward function. For this scheduling problem, the action space would be 'schedule' or 'reschedule' for each order $j = 1, ..., n$; the state space would be 2^j, subject to resource constraints; and the reward function would be the opportunity cost of each order to the business. Related but far more advanced approaches include Deep Reinforcement Learning as explored in Afshar et al. (2020). Dynamic programming is a recursive method that inductively solves for the optimal result, but can be very computationally expensive due to a large number of iterations, and which is explained in Martello and Toth (1984).

3. Methodology

This section describes the linear programming formulation used to solve the scheduling problem.

3.1. Notation

Indices and sets

$r =$ resource	$r = 1, ..., r_{\max}$
$t_r =$ time	$t = 1, ..., t_{r\max}$
$i = (r, t_r) =$ resource-time pair	$i = 1, ..., m$
$j =$ unscheduled orders	$j = 1, ..., n$

where $r_{\max}$ is the number of resources, and $t_{r\max}$ is the number of time periods for resource r. Further, let the index set i be partitioned into three groups $\{i_P, i_C, i_D\}$ corresponding to the product-location-date, lane-date and dock-time resources respectively.

Parameters

w_{ij} number of units of resource i that order j needs in order to be fulfilled

b_i total capacity (or 'budget') of resource i

v_j relative importance of needing to schedule order j vs other orders.

Note that if $v_j = 1$ for all $j = 1, \ldots, m$, then we would be maximizing the number of scheduled orders, or equivalently, minimizing the number of orders that need to be rescheduled. In the algorithm's implementation, we will take v_j as the opportunity cost of rescheduling rather than scheduling order j on its original date.

Decision variables

x_{ij} binary - if order j will be assigned w_{ij} units of resource i (1) or none (0)

y_j binary - if order j has all the resources required for fulfillment (1) or not (0)

where for y_j, the resources required would be all the product, carrier and dock resources for order j.

3.2. Formulation

$$\text{Maximize} \quad \sum_{j=1}^{n} v_j y_j \tag{1}$$

$$\text{Subject to} \quad \sum_{j=1}^{n} w_{ij} x_{ij} \leq b_i \qquad i \in \{i_C, i_D\} \tag{2}$$

$$\sum_{i=(r,1)}^{(r,t_r)} \sum_{j=1}^{n} w_{ij} x_{ij} \leq \min_{t'_r \geq t_r} b_{(r,t'_r)} \qquad i = (r, t_r) \in \{i_P\} \tag{3}$$

$$\sum_{i=1}^{m} x_{ij} \geq y_j \cdot \sum_{i=1}^{m} \mathbf{1}_{w_{ij}>0} \qquad j = 1, ..., n \tag{4}$$

$$x_{ij} \leq \mathbf{1}_{w_{ij}>0} \qquad i = 1, ..., m, j = 1, ..., n \tag{5}$$

$$x_{ij}, y_{ij} \in \{0, 1\} \qquad i = 1, ..., m, j = 1, ..., n \tag{6}$$

Eq. 1 is the objective function as the weighted sum of the business value of orders to be scheduled.

Eq. 2 gives the constraints for carrier and dock availability.

Eq. 3 gives the constraints for product availability, while taking the time of the order into account. Specifically, it sums up all the allocated availability (i.e., of orders that the linear program will mark as 'scheduled') up to the current time t_r, and ensures that it does not exceed the available inventory at any future point $t'_r \geq t_r$. This is because the available inventory can go up and down in the future due to previously-scheduled orders and new production at future time points, but

should never go below zero as a result of deciding to schedule unscheduled orders at an earlier time point.

Eq. 4 ensures that an order will be marked to be scheduled (i.e., $y_j = 1$) if and only if all the resources required for that order will be allocated to that order. The function $\mathbf{1}_X$ is the indicator function, which is 1 if the condition X is true and 0 otherwise. Note that this constraint, together with the objective function to maximize the weighted sum of y_j's, correctly incentivizes filling all resources ($x_{ij} = 1, \forall i$) for a specific order j, or none at all ($x_{ij} = 0, \forall i$).

Eq. 5 ensures that a resource i cannot be allocated to order j unless it needs that resource, i.e., $w_{ij} > 0$. This condition is not necessary if $\{x_{ij}\}$ is only defined for the indices at which $w_{ij} > 0$, but would be needed if a sparse matrix is used in i-j space.

Eq. 6 indicates that this is a binary integer linear program.

4. Results

The proposed algorithm was implemented at a large US CPG manufacturer that had multiple carrier relationships. The values v_j were challenging to calculate tractably or understand easily, and so some simple heuristics were used to approximate the importance of each order for this. The manufacturer's status quo method was to schedule orders based on delivery date (ascending), while having no visibility on product, carrier or dock availability. The algorithm was implemented to optimally allocate all resources, and the orders were then sequenced, first on delivery date (ascending), then on the business value of the order (descending), with the under-resourced orders flagged and recommended to be rescheduled. This reduced unnecessary order reschedules by about 7%, which translated to a large dollar amount for this particular manufacturer due to its scale.

5. Conclusion

The algorithm used in this paper was able to not only provide a view of resource availability, but also optimally calculate how resources should be distributed to competing orders. The result of a 7% improvement in reduced rescheduling suggests that implementing this method can yield significant savings and improved efficiencies, especially for large CPG companies. Further, the particular algorithm described can be flexibly extended to additional and different types of constrained resources.

There are several opportunities for further research to extend this model. One is to take SLA into account, by adding SLA penalties to the objective function, and customer-level constraints to the model, since SLA's are at the customer level. A second opportunity is, in addition to specifying which orders to reschedule, to also prescribe the optimal date and pickup location for those orders. This may also involve specifying penalties in the objective function that are based on the number of days between the original desired date and prescribed rescheduled date.

References

Afshar RR, Zhang Y, Firat M, Kaymak U (2020) A State Aggregation Approach for Solving Knapsack Problem with Deep Reinforcement Learning URL http://arxiv.org/abs/2004.12117.

Assi M, Haraty RA (2018) A Survey of the Knapsack Problem. *2018 International Arab Conference on Information Technology (ACIT)*, 1–6 (IEEE), ISBN 978-1-7281-0385-3, URL http://dx.doi.org/10.1109/ACIT.2018.8672677.

Chekuri C, Khanna S (2005) A Polynomial Time Approximation Scheme for the Multiple Knapsack Problem. *SIAM Journal on Computing* 35(3):713–728, ISSN 0097-5397, URL http://dx.doi.org/10.1137/S0097539700382820.

Fréville A, Hanafi S (2005) The Multidimensional 0-1 Knapsack Problem—Bounds and Computational Aspects. *Annals of Operations Research* 139(1):195–227, ISSN 0254-5330, URL http://dx.doi.org/10.1007/s10479-005-3448-8.

Johnston RE, Khan LR (1995) Bounds for nested knapsack problems. *European Journal of Operational Research* 81(1):154–165, ISSN 03772217, URL http://dx.doi.org/10.1016/0377-2217(93)E0211-F.

Martello S, Toth P (1984) A Mixture of Dynamic Programming and Branch-and-Bound for the Subset-Sum Problem. *Management Science* 30(6):765–771, ISSN 0025-1909, URL http://dx.doi.org/10.1287/mnsc.30.6.765.