ERASMUS UNIVERSITY ROTTERDAM
Erasmus School of Economics

Supply Chain Management with Online Customer Selection

Bachelor Thesis

Econometrie en Operationele Research

Author:
N.M. van der Zon

Student ID:
387441

Supervisor:
W. van den Heuvel

Second Assessor:
R.B.O. Kerkkamp

July 3, 2016

Abstract

In regular supply chain management models, it is often the case that all the customers that have to be served are known in advance. We will consider a more realistic case by looking at models with online customer selection, where customers arrive sequentially, and after each arrival one has to immediately decide whether or not to accept the customer. This means that these selection decisions, which are done by what we call an online algorithm, have to be made with no information available about any future arrivals. We will consider two online algorithms for three different supply chain models with online customer selection: the joint replenishment problem, the economic lot sizing problem and the traveling salesman problem. In addition to that, we perform computational experiments to evaluate the performance of these algorithms in several scenarios. These experiments show that the cost incurred by the online algorithm is very close to the optimal cost incurred when one would have full knowledge of the customer arrivals in advance.

Contents

1 Introduction

In most traditional supply chain models, the goal is to optimize the production process given an exogenous set of customers. Recent research showed that this set of customers does not have to be completely exogenous, because the supplier can also influence the demand side through their decisions. In particular, the supplier can choose which customers they will serve, so that the demand optimally matches the production capabilities (Geunes et al., 2011). We call this *customer selection*.

One can distinguish between *offline* customer selection and *online* customer selection. In problems with offline customer selection, the information on all of the customers is known beforehand. This means that all information can be used in making the decision which customers to accept or to reject. In reality, this is often not the case. Instead, we can consider online customer selection, where all the customers arrive sequentially, and the supplier immediately has to make the decision whether or not to serve a customer after he arrives. These selection decisions are made by an *online algorithm*. If a customer is rejected, the supplier must pay a certain rejection cost. For the accepted customers, production costs will be incurred. We can see how an online algorithm performs by dividing the total cost incurred by the online algorithm by the optimal total cost of the offline problem, which gives us a *performance ratio*. In some cases, we can also derive an upper bound on the performance ratios of an online algorithm, which we call a *competitive ratio*. This means that the total costs are always within a constant factor of the optimal costs of the offline solution for any sequence of customers.

The aim of this thesis is to investigate the performance of two online algorithms on three different supply chain models with online customer selection. This will be based on the work of Elmachtoub and Levi (2016), who consider two online algorithms and test them on three different supply chain models. We will verify the results of the algorithms on two of these models, namely the *joint replenishment* (JR) problem, and its special case, the *economic lot sizing* (ELS) problem. In addition to that, we will extend the work of Elmachtoub and Levi (2016). We look at ways to improve the performance of the online algorithms by using a so-called scaling factor. Furthermore, we consider an extension of the JR problem in which there are capacities on the number of products per order. We also derive competitive ratios for both algorithms for this extended problem. Finally, we will also implement one of the two algorithms for the *traveling salesman* (TS) problem.

The rest of this thesis is organized as follows. Section 2 provides an overview of the existing literature on this subject. In section 3, we present the general model for problems with online customer selection. Section 4 presents the two online algorithms we will use, and describes the way to measure their performance in more detail. In sections 5 and 6, we show how to apply the general model to the JR, ELS and TS problems. Section 7 shows the results of the computational experiments we performed. We also compare the results for the JR and ELS problems to the results of Elmachtoub and Levi (2016). Finally, section 8 provides a conclusion and gives some ideas for future research.

2 Literature Review

Over the past 20 years, there has been a lot of interest in models with customer selection. Slotnick (2011) gives a survey of order acceptance problems. In the category of offline selection problems, Geunes et al. (2011) consider several supply chain problems (including the JR problem) with *offline market selection*, where a market is defined as a sequence of demands requested by customers over time. Customer selection is a special case of

this, where each customer has only one demand request. The ELS problem with offline customer selection is studied by Geunes et al. (2006), where they provide polynomial time solution approaches for both capacitated and uncapacitated versions.

Most of the research on online optimization has been done in the area of computer science, and only recently the concept of online optimization has been used to study models in operations research (Elmachtoub and Levi, 2016). For an introduction to online algorithms, see Karp (1992) and Albers (2003). As an application, Epstein et al. (2002) consider an online algorithm for job scheduling. All of these papers consider applications in computer science, but most of the concepts can be applied to online supply chain models as well.

Regarding the online ELS and JR problems that we will consider, Van den Heuvel and Wagelmans (2010) study the worst-case performance of online heuristics for the ELS problem, and provide a lower bound on the competitive ratio of any online heuristic for this problem. For the JR problem, Buchbinder et al. (2013) provide an online primal-dual algorithm, using the linear programming relaxation of the offline JR problem.

There is also a lot of literature on online optimization for vehicle routing problems (see the survey by Jaillet and Wagner (2008)). Jaillet and Lu (2014) consider several variants of the TS problem with online customer selection, including the online price-collecting traveling salesman problem, which is studied by Ausiello et al. (2008) as well. They distinguish between a basic version, in which customers can be accepted or rejected any time after their arrival, and the real-time version where this selection decision has to happen immediately after arrival. The real-time version is the most similar to Elmachtoub and Levi (2016), however it is a bit more dynamic. In Ascheuer et al. (2000), a more general case of the online TS problem, called the online dial-a-ride problem, is studied. In this problem, a server has to transport objects from their source location to their destination. New transportation requests come in while the server is busy transporting the objects, and these requests can either be accepted or rejected. In their paper, three different online algorithms are presented for this problem.

Finally, there are a lot of other applications of online optimization in the area of operations research. In Keskinocak et al. (2001), an online version of a scheduling problem with lead time quotations is considered. They give several online algorithms and derive competitive ratios. Another application is the online facility location problem studied by Fotakis (2008). He also provides a lower and upper bound on the performance that depends on the number of customers. Elmachtoub and Levi (2016) also consider the same online facility location problem, but use a different approach.

3 Problem Definition

We will now present the general model for problems with online customer selection as given by Elmachtoub and Levi (2016). This model will be used in the rest of the thesis, where we apply it to the JR, ELS and TS problems with online customer selection.

When we consider problems with online customer selection, we divide the problem into two different phases: the *selection phase* and the *production phase*. The objective is then to minimize the total cost, which is the sum of the rejection costs from the selection phase and the production costs from the production phase.

3.1 Selection Phase

The first phase is the *selection phase*, where we sequentially select which customers to accept or reject. In each *stage k* of the selection phase, a customer k arrives with certain requirements I_k. After the customer arrives, the supplier immediately has to decide whether or not to serve this customer. For each rejected customer, a cost of r_k is incurred. This rejection cost often depends on the specific requirements of a customer.

The decisions in the selection phase are made by the online algorithm. Let N be the number of customers that arrived, which we do not know beforehand, so that $\mathcal{U} = \{1, \ldots, N\}$ is the full set of customers. We will also refer to this, together with the requirements of the customers, as a customer sequence. The first k customers are in the set $\mathcal{U}_k = \{1, \ldots, k\}$. Then, by using the online algorithm, we obtain in each stage k the sets $\mathcal{A}_k$ which contains the accepted customers up to then, and $\mathcal{R}_k$ which contains the rejected customers. Note that $\mathcal{A}_{k-1} \subseteq \mathcal{A}_k$ and $\mathcal{R}_{k-1} \subseteq \mathcal{R}_k$ for all k. This means previous selection decisions are fixed.

After all customers have arrived, we finish with the sets $\mathcal{A}$ and $\mathcal{R}$. For convenience, the subscripts are dropped when referring to the final stage N, such that $\mathcal{A} = \mathcal{A}_N$ and $\mathcal{R} = \mathcal{R}_N$. For the customers in $\mathcal{R}$ a rejection cost of $R(\mathcal{R}) = R(\mathcal{U} \setminus \mathcal{A}) = \Sigma_{k \in \mathcal{R}} r_k$ is paid.

3.2 Production Phase

The second phase is the *production phase*, where we have to serve the customers in $\mathcal{A}$ at minimum costs while keeping each of their requirements in mind. Let $\mathcal{Q}$ be the set of the production options that are available, e.g., a set of possible order dates. Then for any $Q \subseteq \mathcal{Q}$ and $T \subseteq \mathcal{U}$, we denote $P(Q, T)$ as the minimum possible production costs to serve all customers in T using only the production options in Q. The method to find the minimum possible costs varies for each problem, and we will show this later on in sections 5 and 6 when we look at the implementation of the JR and TS problems.

Again, for convenience, we often drop the production options input in the cost function P if we use the entire set $\mathcal{Q}$, i.e., $P(\mathcal{Q}, \cdot) = P(\cdot)$.

3.3 Offline Comparison

As we will see, the solution to the online customer selection problem is often compared to the solution of its offline variant. In the *offline* customer selection problem, all information $I_1, \ldots, I_N$ and $r_1, \ldots, r_N$ of the N stages is known a priori. We can then define the offline problem as:

$$OPT(\mathcal{Q}, \mathcal{U}) = \min_{\mathcal{A} \subseteq \mathcal{U}} \{P(\mathcal{Q}, \mathcal{A}) + R(\mathcal{U} \setminus \mathcal{A})\}.$$

Denote the optimal partition of customers in stage k as $\mathcal{A}_k^*$ and $\mathcal{R}_k^*$. Note that in the offline customer selection problem, previously made selection decisions may change in later stages. This means that a customer who was accepted in stage $k-1$ might be rejected in stage k.

4 Online Algorithms

Elmachtoub and Levi (2016) present two online algorithms. We will use these two algorithms to solve the JR and ELS problems with online customer selection. For the TS problem, we will only use one of these two algorithms.

4.1 Copycat Algorithm

The first algorithm is the *Copycat algorithm*. It is called this way because the selection decision in each stage is simply copied from an optimal solution to the offline problem.

Copycat Algorithm

Accept current customer k if and only if $k \in \mathcal{A}_k^*$, where $\mathcal{A}_k^*$ is obtained by solving the offline customer selection problem $OPT(\mathcal{Q}, \mathcal{U}_k)$.

The Copycat algorithm requires an exact solution to the offline customer selection problem for every customer that arrives. This can be a huge disadvantage if the offline problem is NP-hard, such as for the JR problem (Arkin et al., 1989). For this reason, another algorithm is presented that makes the selection phase efficiently computable.

4.2 StablePair Algorithm

The *StablePair algorithm* can solve the selection phase in polynomial time for many problems (including the JR problem). Before we give the description of the StablePair algorithm, we first have to introduce the concept of a *stable pair*. For every subset of production options $Q \subseteq \mathcal{Q}$ and customers $T \subseteq \mathcal{U}$, we call (Q, T) a *stable pair* if there exists an optimal solution to the offline problem OPT(Q,T) where all customers in T are accepted.

StablePair Algorithm

Accept current customer k if and only if there exists a stable pair $(Q, T) \subseteq (\mathcal{Q}, \mathcal{U}_k)$ such that $k \in T$.

4.3 Measuring Performance

In order to evaluate the performance of an online algorithm, we often compare the online solution to the optimal offline solution. Define the total cost of the optimal offline solution as $C^*(\mathcal{U}) = P(\mathcal{A}^*) + R(\mathcal{R}^*)$. We can then use these offline costs, together with the online costs $C(\mathcal{U}) = P(\mathcal{A}) + R(\mathcal{R})$, to evaluate the performance of an online algorithm using *competitive analysis* (Albers, 2003). If the total cost of the online solution is no more than c times higher than the total cost of the optimal offline solution, for any type of input, we say that the algorithm is *c-competitive* or, equivalently, has a *competitive ratio* of c. This provides us a worst-case performance measure.

In Elmachtoub and Levi (2016), the following two theorems are proven. These theorems allow us to find competitive ratios for both Copycat and StablePair if we have an upper bound on the production cost.

Theorem 1

Let β be a positive scalar. If $P(\mathcal{A}) \leq \beta C^*(\mathcal{U})$, then both Copycat and StablePair are $(\beta + 1)$-competitive.

For the StablePair algorithm, we can obtain an even better competitive ratio if we can bound the production cost in a more specific way.

Theorem 2

Let β and γ be a positive scalar. If $P(\mathcal{A}) \leq \beta P(\mathcal{A}^*) + \gamma R(\mathcal{R}^*) + R(\mathcal{A} \cap \mathcal{R}^*)$, then StablePair is $\max(\beta + 1, \gamma + 1)$-competitive.

These theorems were used to find the competitive ratios of Copycat and StablePair for the JR and ELS problems in Elmachtoub and Levi (2016). We will use them later on as well to find the competitive ratios of the JR problem with soft capacities.

Furthermore, we can also compute the *performance ratio* $C(\mathcal{U}_k)/C^*(\mathcal{U}_k)$ in each stage for a particular customer sequence. The lower the performance ratio, the better the performance. This gives us the empirical performance in each stage of a sequence, and can be compared with the theoretical upper bound given by the competitive ratio.

4.4 Improving Performance via Scaling

The performance of the StablePair algorithm can be improved even further by using a scaling technique. For a certain *scaling factor* ϕ_k, we multiply the rejection costs of customer k by ϕ_k and then apply the StablePair algorithm to the scaled input. For the problems we consider, a higher scaling factor gives us better performance ratios in later stages, at the cost of a bit worse performance in the first few stages. This means that if a lot of customers are expected, a higher scaling factor should be chosen.

Elmachtoub and Levi (2016) only look at a constant scaling factor $\phi_k = 2$ for all customers k. However, there is no clear reason given why they chose this value. That is why we will consider two different classes of scaling factors in order to find the scaling factor for which the mean performance ratio is the lowest. Note that this mean performance ratio highly depends on the input data, i.e. the customer arrivals. Therefore, we want a scaling factor that has the lowest mean performance ratio over multiple sequences of customer arrivals, where this is computed by taking the average of all performance ratios in all stages of all customer sequences. This means that for a given customer sequence, the mean performance ratio in that particular sequence does not have to be minimal.

The first class of scaling factors we will look at are the *constant* scaling factors, where $\phi_k(\alpha) = \alpha$ for all customers k. Next to that, we will also consider the class of *linear* scaling factors, i.e., $\phi_k(\alpha, \beta) = \alpha + \beta k$. Of course, the constant scaling factor is a special case of this, where β equals 0. In both cases, $\phi_k(\cdot)$ is a function, which we will more generally denote as $\phi_k(\mathbf{x})$, where $\mathbf{x}$ is a vector of scalar parameters, e.g., $\mathbf{x} = \begin{bmatrix}\alpha & \beta\end{bmatrix}$ for the linear case. The dimension of $\mathbf{x}$ depends on the number of parameters in the scaling factor.

Define $f(\mathbf{x})$ as the mean performance ratio obtained by executing the StablePair algorithm with a scaling factor $\phi_k(\mathbf{x})$ on several customer sequences. The problem we then have to solve is minimizing $f(\mathbf{x})$ with respect to $\mathbf{x}$. This can be done by using any direct search method. We will use the *Nelder-Mead algorithm* as done in Lagarias et al. (1998), which resolved some ambiguities of the original version by Nelder and Mead (1965). Furthermore, we will directly fill in the standard coefficients in the algorithm to avoid excessive notation.

The algorithm requires $n + 1$ initial points $\mathbf{x}_1, \ldots, \mathbf{x}_{n+1}$, where n is the number of parameters in the scaling factor. We call the shape that is formed by these points a simplex. With these points, we iteratively execute several steps. In each step, we compute a new point by transforming the original $n + 1$ points in a certain way. The function value in this new point is then compared to function values of other points. Depending on the outcome of this comparison, we either accept the new point by replacing the worst point with it, or we continue to the next step and perform a different transformation. You can visualize this as the simplex 'moving' towards the minimum. We stop when the standard deviation of the function values of the $n + 1$ points is below a certain threshold.

For all of these steps, in case of equal function values when a new point is accepted, we give the new point the higher index. Also note that this algorithm does not guarantee optimality, which means we can get stuck in a local minimum. We can be more certain

about our result by performing this algorithm multiple times using different initial points.

Nelder-Mead algorithm

Step 1. Order
Order and relabel the $n+1$ points, such that $f(\mathbf{x}_1) \leq \ldots \leq f(\mathbf{x}_{n+1})$. Compute the mean of the first n points (all except $\mathbf{x}_{n+1}$), i.e., $\bar{\mathbf{x}} = \frac{1}{n}\Sigma_{i=1}^{n}\mathbf{x}_i$. If the sample standard deviation of the function values of the $n+1$ points is below 10^{-8}, so that the function values are all very close to each other, STOP.

Step 2. Reflect
Compute the *reflection* point $\mathbf{x}_r = \bar{\mathbf{x}} + (\bar{\mathbf{x}} - \mathbf{x}_{n+1})$. If $f(\mathbf{x}_1) \leq f(\mathbf{x}_r) < f(\mathbf{x}_n)$, replace point $\mathbf{x}_{n+1}$ with $\mathbf{x}_r$ and go back to Step 1.

Step 3. Expand
If $f(\mathbf{x}_r) < f(\mathbf{x}_1)$, compute the *expansion* point $\mathbf{x}_e = \bar{\mathbf{x}} + 2(\mathbf{x}_r - \bar{\mathbf{x}})$. If $f(\mathbf{x}_e) < f(\mathbf{x}_r)$, replace point $\mathbf{x}_{n+1}$ with $\mathbf{x}_e$ and go back to Step 1. Otherwise, if $f(\mathbf{x}_e) \geq f(\mathbf{x}_r)$, replace point $\mathbf{x}_{n+1}$ with $\mathbf{x}_r$ and go back to Step 1.

Step 4. Contract
If $f(\mathbf{x}_r) \geq f(\mathbf{x}_n)$, perform either an *outside* or *inside* contraction.

a. Outside If $f(\mathbf{x}_n) \leq f(\mathbf{x}_r) < f(\mathbf{x}_{n+1})$, compute $\mathbf{x}_c = \bar{\mathbf{x}} + 0.5(\mathbf{x}_r - \bar{\mathbf{x}})$. If $f(\mathbf{x}_c) \leq f(\mathbf{x}_r)$, replace point $\mathbf{x}_{n+1}$ with $\mathbf{x}_c$ and go back to Step 1.

b. Inside If $f(\mathbf{x}_r) \geq f(\mathbf{x}_{n+1})$, compute $\mathbf{x}_{cc} = \bar{\mathbf{x}} - 0.5(\bar{\mathbf{x}} - \mathbf{x}_{n+1})$. If $f(\mathbf{x}_{cc}) < f(\mathbf{x}_{n+1})$, replace point $\mathbf{x}_{n+1}$ with $\mathbf{x}_{cc}$ and go back to Step 1.

Step 5. Shrink
For all $i = 2, \ldots, n+1$, replace the point $\mathbf{x}_i$ with $\mathbf{v}_i = \mathbf{x}_1 + 0.5(\mathbf{x}_i - \mathbf{x}_1)$, so that the (unordered) points are $\mathbf{x}_1, \mathbf{v}_2, \ldots \mathbf{v}_{n+1}$. Go back to Step 1.

The result of this algorithm will be $\mathbf{x}_1$, which contains the parameters of the scaling factor with the lowest mean performance ratio.

5 Joint Replenishment Problem

The *joint replenishment problem* we consider is a discrete time inventory model with multiple item types. The goal is to optimally serve customer demands by choosing when to place production orders over a certain planning horizon. Each customer has a demand quantity, item type and due date. For every order that is placed, a *joint setup cost* of K_0 is occurred. In addition to that, for every order of item type $i \in \{1, \ldots, M\}$ a *type setup cost* of K_i has to be paid. Next to that, there also is a *holding cost* of h per period per unit. This holding cost does not depend on the item type. The production options $\mathcal{Q}$ are the potential order dates. The objective is then to minimize the production cost, which is the sum of the setup costs and the holding costs.

In the joint replenishment problem with online customer selection, we have that in each stage k, a customer arrives with requirements $I_k = (d_k, i_k, t_k)$. Here, d_k is the demand quantity, i_k is the item type and t_k is the due date of customer k. The rejection cost of the customer depends on the quantity that is demanded, such that $r_k = rd_k$, where r is the rejection cost per unit. The goal is now to minimize the rejection cost plus the production cost, which we will solve, as mentioned before, in two phases.

5.1 Selection Phase

5.1.1 Copycat Implementation

The Copycat algorithm requires in each stage the optimal solution to the offline problem. Geunes et al. (2011) give an integer programming formulation to solve the JR problem with offline market selection. Customer selection is a special case of this, where each market consists of only one demand request. This means we can use the same formulation to solve the offline customer selection problem.

Let the discrete planning horizon be $\{1, \ldots, T\}$. Then, for every customer $j \in \mathcal{U}$ and period $s \in \{1, \ldots, T\}$, define the parameter H_s^j as the total holding cost for satisfying the demand of customer j from a production order placed in period s:

$$H_s^j = \begin{cases} h(t_j - s)d_j & \text{if } s \leq t_j \\ \infty & \text{otherwise.} \end{cases}$$

Let y_{0s} be a binary variable that is equal to 1 if a production order (of any type) is placed in period s. In the same way, define y_{is} as a binary variable that equals 1 if an order of type i is placed in period s. These variables are required to keep track of the setup costs. Next to that, let x_s^j be the fraction of the demand of customer j that is satisfied by a production order in period s, where x_s^j is only defined for $s \leq t_j$. At last, let z_j be a binary variable that equals 0 if customer j is selected, and equals 1 otherwise.

For every stage k, we have to solve the following integer program to obtain the optimal offline solution:

$$\begin{aligned}
\text{minimize} \quad & \sum_{s=1}^{T} K_0 y_{0s} + \sum_{i=1}^{M} \sum_{s=1}^{T} K_i y_{is} + \sum_{j=1}^{k} \sum_{s=1}^{t_j} H_s^j x_s^j + \sum_{j=1}^{k} r_j z_j \\
\text{subject to} \quad & \sum_{s=1}^{t_j} x_s^j = 1 - z_j \qquad j = 1, \ldots, k; && (1) \\
& x_s^j \leq y_{is} \qquad j = 1, \ldots, k; \quad i = i_j; \quad s = 1, \ldots, t_j; && (2) \\
& x_s^j \leq y_{0s} \qquad j = 1, \ldots, k; \quad s = 1, \ldots, t_j; && (3) \\
& x_s^j \geq 0 \qquad j = 1, \ldots, k; \quad s = 1, \ldots, t_j; \\
& y_{is} \in \{0, 1\} \qquad i = 0, \ldots, M; \quad s = 1, \ldots, T; \\
& z_j \in \{0, 1\} \qquad j = 1, \ldots, k.
\end{aligned}$$

The objective function is the sum of the setup costs, the holding costs and the rejection costs. Constraints (1) make sure that all customers that are selected have their demands satisfied on time. Constraints (2) ensure that if an order of type i is placed, the corresponding setup cost is paid. At last, constraints (3) ensure that the joint setup cost is paid for every order that is placed.

After solving this integer program, we look at the optimal value of z_k. If this value is equal to 0, then customer k is accepted by the Copycat algorithm. Otherwise, the customer will be rejected.

5.1.2 StablePair Implementation

As described in section 4.2, the StablePair algorithm accepts customer k if there exists a stable pair (Q, T) for which $k \in T$. Therefore, we will generate stable pairs until we find one where this condition holds. This algorithm provides us an efficient way to solve the selection phase, because the set of stable pairs we have to check can be generated in polynomial time. To check if a pair (Q, T) is a stable pair, we have to check four properties, as given by Elmachtoub and Levi (2016).

1. Q consists of one order date, i.e., $Q = \{t\}$ for some period t.
2. If T contains customers of type i, then it contains all type i customers with a due date in $[t, t + r/h]$. Let $T^i = \{j \in \mathcal{U}_k : t_j \in [t, t + r/h] \text{ and } i_j = i\}$, then if T contains customers with item types in $\mathcal{F}$, we have $T = \cup_{i \in \mathcal{F}} T^i$.
3. The types of customers included in T are those who can pay for the item setup cost and the holding costs. This means that $\mathcal{F} = \{i : R(T^i) \geq K_i + \Sigma_{j \in T^i} h(t_j - t)d_j\}$.
4. The total rejection costs of T are at least the production costs of serving T from period t.

We can then use the following algorithm in each stage k to generate stable pairs and to determine whether customer k is accepted or not.

Algorithm StablePair for JR problem in stage k

```
function STABLEPAIR(k)
    for t ← 1, ..., t_k do
        T^i ← {j ∈ U_k : t_j ∈ [t, t + r/h] and i_j = i}
        F ← {i : R(T^i) ≥ K_i + Σ_{j∈T^i} h(t_j − t)d_j}
        T ← ∅
        for i ∈ F do
            T ← T ∪ T^i
        end for
        if k ∈ T and R(T) ≥ K_0 + Σ_{i∈F}(K_i + Σ_{j∈T^i} h(t_j − t)d_j) then
            return accept customer k
        end if
    end for
    return reject customer k
end function
```

5.2 Production Phase

After the selection phase is solved, we have to satisfy the demands of the accepted customers at minimum costs. Because the JR problem is NP-hard (Arkin et al., 1989), we will use the 1.8-approximation algorithm from Levi et al. (2008). This algorithm is based on rounding an LP solution. The formulation we will use comes from Levi et al. (2006) and is similar to the one used in the Copycat algorithm, except there are no customer selection decisions being made this time.

When using a c-approximation algorithm to solve the production phase, the competitive ratio is multiplied by c as well. So in this case, the competitive ratios will be 1.8 times

higher. If, for that reason, an exact solution to the production phase is preferred, the same formulation can be used if the decision variables are made binary. The rounding step can be skipped as well in that case.

Formulation

Define the parameter d_{it} as the total demand, which is the sum over all accepted customer demands, for type i products in period t. Let y_{0s} be a variable indicating whether a production order of *any* type is placed in period s. Similarly, we define y_{is} as a variable that indicates whether an order of type i is placed in period s. Finally, let x_{ist} indicate whether the demand d_{it} is satisfied by a production order in period s:

$$
\begin{aligned}
\text{minimize} \quad & \sum_{s=1}^{T} K_0 y_{0s} + \sum_{i=1}^{M}\sum_{s=1}^{T} K_i y_{is} + \sum_{i=1}^{M}\sum_{t=1}^{T}\sum_{s=1}^{t} h(t-s) d_{it} x_{ist} \\
\text{subject to} \quad & \sum_{s=1}^{t} x_{ist} = 1 \qquad \forall (i,t) \in \{(i,t) : d_{it} > 0\} && (1) \\
& x_{ist} \leq y_{is} \qquad i = 1,\dots,M; \quad t = 1,\dots,T; \quad s = 1,\dots,t; && (2) \\
& x_{ist} \leq y_{0s} \qquad i = 1,\dots,M; \quad t = 1,\dots,T; \quad s = 1,\dots,t; && (3) \\
& x_{ist} \geq 0 \qquad i = 1,\dots,M; \quad s = 1,\dots,T; \quad t = s,\dots,T; \\
& y_{is} \geq 0 \qquad i = 0,\dots,M; \quad s = 1,\dots,T.
\end{aligned}
$$

The objective function is the total production cost, which is the sum of the setup and holding costs. Constraints (1) make sure that all demands are satisfied on time. Constraints (2) ensure that if a production order of type i is placed, the corresponding setup cost is paid. At last, constraints (3) ensure that the joint setup cost is paid for every production order that is placed.

The optimal solution is then used in the rounding step, where we will obtain a feasible solution such that the total production cost is at most 1.8 times the optimal cost.

Rounding Step

The rounding step works as follows. We first determine the periods in which we should place production orders, for which we use the deterministic variant of the *shift procedure* in Levi et al. (2008). Then, once these periods are fixed, the problem decomposes into M single-item lot sizing problems, for which we can use the dynamic programming algorithm from Wagner and Whitin (1958) to solve it.

The input to the shift procedure is a *step parameter* $c \in (0,1]$. We also use the optimal solution to the linear program, which we denote as $\hat{x}_{ist}$, $\hat{y}_{0s}$ and $\hat{y}_{is}$. With this solution, we can construct several intervals based on the values of $\hat{y}_{0s}$. For each period $t = 1,\dots,T$, let $\hat{Y}_{0t} = (\Sigma_{s=1}^{t-1}\hat{y}_{0s}, \Sigma_{s=1}^{t}\hat{y}_{0s}]$, so that the interval length is $\hat{y}_{0t}$. Furthermore, let W be the smallest integer multiple of c that is greater than $\Sigma_{s=1}^{T}\hat{y}_{0s}$, i.e., $W = \lceil \frac{1}{c}\Sigma_{s=1}^{T}\hat{y}_{0s} \rceil$.

By introducing a new parameter $\alpha \in (0,1]$, which we call the *shift parameter*, we can construct the following set of *shift points*: $\{\alpha + cw : w = 0,\dots,W-1\}$. These shift points are used to determine in which periods the orders are placed. For each period $m = 1,\dots,T$, we place an order in period m if there is at least one shift point within the interval $\hat{Y}_{0m} = (\Sigma_{s=1}^{m-1}\hat{y}_{0s}, \Sigma_{s=1}^{m}\hat{y}_{0s}]$.

We can use the dynamic programming algorithm from Wagner and Whitin (1958) to find the solution for a set of order points. Let $V(i,t)$ be the optimal cost for the subproblem of type i for periods 1 to t, with $V(i,0) = 0$ as initial value. In addition, define $\mathcal{O}(i,t)$ as the set of order periods for type i that are before or at time t. Note that the order points found by the shift procedure are production orders for *any* type, so that means that if there is no need to place an order for type i in one of these periods, then the period should not be considered for type i and therefore not included in $\mathcal{O}(i,t)$. For example, if we have an order point at period s, and there is no demand for type i in any of the periods p with $s \leq p \leq t$, then this order point should be excluded from $\mathcal{O}(i,t)$. We can then find the solution by solving the following equation for each type $i = 1,\ldots,M$ and each period $t = 1,\ldots,T$:

$$V(i,t) = \min_{s \in \mathcal{O}(i,t)} \{V(i,s-1) + K_i + \sum_{p=s}^{t} h(p-s)d_{ip}\}.$$

It is easily verified that there are only a polynomial number of values of α that generate distinct sets of order periods. The reason for this is that we only have to look at sequences of shift points where (at least) one of them is at the right boundary of an interval $\hat{Y}_{0m}$. By iterating over these values of α, we can calculate the cost of the optimal solution using only the associated order periods, and then choose the solution with the minimum cost. For the step parameter c, any value between 0 and 1 can be chosen, but we took $c = 1$ for our analysis.

5.3 Including Soft Capacities

In order to make the JR problem a bit more realistic, one can consider the extension where there are *soft capacities*. This means that there is an upper bound L_{is} to the number of units of type i that can be produced in an order placed in period s, but there is no limit on the number of production orders that can be placed in any time period. Thus, no more than L_{is} units of demand of type i can be served from a particular order in period s, and an extra production order has to be placed if you have to serve more demands. Next to that, we also have a maximum capacity L_{0s} on the total number of units (of all types) that can be produced in a production order placed in period s.

We will solve this extended problem with the Copycat algorithm, for which we adapt both the formulation for the offline customer selection problem and the formulation used to solve the production phase. Note that we can not use the approximation algorithm we used for the production phase. There exists a 3.6-approximation algorithm for the problem with soft capacities, as described in Levi et al. (2008), but we will solve the production phase exactly this time. The formulation for the offline problem is modified by adding the following two constraints, which come from Geunes et al. (2011), where $\mathcal{U}_{is}^k = \{j \in \mathcal{U}_k : i = i_j,\ s \leq t_j\}$:

$$\sum_{j \in \mathcal{U}_{is}^k} x_s^j d_j \leq y_{is} L_{is} \qquad i = 1,\ldots,M; \quad s = 1,\ldots,T;$$

$$\sum_{i=1}^{M} \sum_{j \in \mathcal{U}_{is}^k} x_s^j d_j \leq y_{0s} L_{0s} \qquad s = 1,\ldots,T.$$

The formulation used to solve the production phase requires a similar modification, where we add the following two constraints:

$$\sum_{t=s}^{T} x_{ist} d_{it} \leq y_{is} L_{is} \qquad i = 1, \ldots, M; \quad s = 1, \ldots, T;$$

$$\sum_{i=1}^{M} \sum_{t=s}^{T} x_{ist} d_{it} \leq y_{0s} L_{0s} \qquad s = 1, \ldots, T.$$

Furthermore, in both formulations, for each $i = 0, \ldots, M$ and $s = 1, \ldots, T$, we change the y_{is} variables from binary to integer.

5.4 Competitive Ratios

Elmachtoub and Levi (2016) prove that the competitive ratio for the JR problem with customer selection for any deterministic online algorithm is at least 2. They also proved, by bounding the production costs, that Copycat and StablePair have competitive ratios of 4 and 3, respectively. This does not include the use of the 1.8-approximation algorithm, in which case these competitive ratios have to be multiplied by 1.8. The competitive ratios are a bit higher than the theoretical lower bound of 2, but keep in mind that a competitive ratio gives a worst-case performance measure. As we will see in section 7.1, both algorithms perform significantly better, even when using the approximation algorithm.

For the problem with soft capacities, we will derive competitive ratios for both the Copycat and StablePair algorithms for the case where $\Sigma_{i=1}^{M} K_i \leq K_0$ and the soft capacities are stable, i.e., L_{is} is constant for all $i = 0, \ldots, M$ and $s = 1, \ldots, T$. If this is the case, then the following lemma gives an upper bound on the production costs that will be incurred. The proof can be found in appendix A.1.

Lemma 1

If the Copycat or StablePair algorithm is used for the JR problem with stable soft capacities, and $\Sigma_{i=1}^{M} K_i \leq K_0$, then $P(\mathcal{A}) \leq 5P(\mathcal{A}^*) + 5R(\mathcal{A} \cap \mathcal{R}^*) \leq 5C^*(\mathcal{U})$.

Combining Lemma 1 with Theorems 1 and 2, we get the following result.

Theorem 3

The Copycat or StablePair algorithm for the JR problem with stable soft capacities, and $\Sigma_{i=1}^{M} K_i \leq K_0$, are both 6-competitive.

6 Traveling Salesman Problem

In the *traveling salesman problem* we consider, we have several customers located in a certain *metric space* $\mathcal{M}$ with a *distance function* $d(\cdot, \cdot)$ and an *origin* o. We assume that the distances are symmetric, such that for any two locations x and y, we have $d(x, y) = d(y, x)$. Next to that, the distances should obey the triangle inequality. The goal is to find a minimum length tour that starts and ends at the origin and visits each of these customers exactly once. Each customer k has a certain demand d_k and is at a certain location $l_k \in \mathcal{M}$, i.e., a point in the metric space. The set of production options $\mathcal{Q}$ consists of all the possible tours that can be made. Furthermore, let p be the cost to travel one distance unit. The objective is then to minimize the production cost, which is p times the total distance of the tour.

In the traveling salesman problem with online customer selection, we have that in each stage k, a customer arrives with requirements $I_k = (l_k, d_k)$. Here, l_k is the location of

customer k in the metric space $\mathcal{M}$ and d_k is his demand. The rejection cost depends on the demand of the customer, such that $r_k = rd_k$, where r is the rejection cost per unit of demand. The goal is now to minimize the rejection cost plus the production cost, which we will solve, just as for the JR problem, in two phases.

6.1 Selection Phase

Because of the problem structure, implementing the StablePair algorithm will be very difficult. Therefore, after showing how to implement the Copycat algorithm, we present an alternative method that solves the selection phase more efficiently than Copycat.

6.1.1 Copycat Implementation

For the Copycat algorithm, we require the optimal solution to the offline problem in each stage. Laporte (1992) presents an integer program that we can use to solve the basic TS problem, i.e. the problem without any customer selection. We can use the optimal solution to this integer program in our procedure to solve the offline problem.

In each stage k, we construct a graph $G^k = (V^k, E^k)$, where $V^k = \{l_1, \ldots, l_k\} \cup \{o\}$ are the vertices and E^k are the edges between all customer locations plus the origin o. For any $i < j$, we denote the edge between the location of customer i and the location of customer j as (i, j). The edge between the origin and a customer i is denoted as (o, i). Let x_{ij} be a binary variable that is equal to 1 if the edge (i, j) is used in the optimal solution. Finally, let $c_{ij} = p \cdot d(l_i, l_j)$ be the cost to travel over the edge (i, j). Note that because of the symmetry, we have $c_{ij} = c_{ji}$.

In order to solve the offline problem in each stage k, we decompose the problem into several subproblems where we solve the TS problem for a subset of the vertices and edges. For each $Z \subseteq \mathcal{U}_k$, define $V_Z \subseteq V^k$ as the set of locations of customers in Z plus the origin o, and let E_Z be the corresponding set of edges between every vertex in V_Z. We then have to solve the following minimization problem:

$$\min_{Z \subseteq \mathcal{U}_k} \{\phi(Z) + \sum_{j \in \mathcal{U}_k : j \notin Z} r_j\}.$$

where $\phi(Z)$ is the optimal objective value of the integer program given below:

$$\begin{aligned}
&\text{minimize} && \sum_{(i,j) \in E_Z} c_{ij} x_{ij} && && (\phi(Z))\\
&\text{subject to} && \sum_{i \in V_Z : (i,j) \in E_Z} x_{ij} + \sum_{i \in V_Z : (j,i) \in E_Z} x_{ji} = 2 && j \in V_Z; && (1)\\
& && \sum_{(i,j) \in E_Z : i \in S, j \notin S} x_{ij} + \sum_{(j,i) \in E_Z : i \in S, j \notin S} x_{ji} \geq 2 && S \subset V_Z;\ 3 \leq |S| \leq \lfloor |V_Z|/2 \rfloor; && (2)\\
& && x_{ij} \in \{0,1\} && (i,j) \in E_Z. &&
\end{aligned}$$

Constraints (1) make sure that all customers are visited exactly once. Constraints (2) ensure that the tour contains no subtours. The objective value represents the optimal production cost for all selected customers, i.e. the customers in Z. Note that this formulation only works when $|V_Z| \geq 3$, i.e. if there are at least two customers that have to be served (plus the origin). If there is only one customer in V_Z, then $\phi(Z)$ is simply twice the distance from this customer to the origin. And trivially, if there are no customers, $\phi(Z)$ will be zero.

After solving the minimization problem, we check if customer k is contained in the optimal Z. If this is the case, Copycat accepts customer k. Otherwise, the customer is rejected.

6.1.2 Approximate Copycat

There are two large disadvantages for the Copycat algorithm. The first one is that the TS problem is NP-hard (Laporte, 1992), which means that computing $\phi(Z)$ is a hard problem and can take a lot of time. Secondly, to solve the minimization problem, we have to iterate over all $Z \subseteq \mathcal{U}_k$. This means that the number of possible subsets will increase exponentially when the number of customers gets larger. Combining these two factors, the Copycat algorithm will quickly not be able to solve the problem in reasonable time anymore. It therefore makes sense to look at alternative ways to solve the selection phase. That is why we will look at a different way to reduce the complexity.

In the implementation of the Copycat algorithm for the JR problem, we were able to include the selection decisions directly in the integer program by adding the z_j variables. If we can do this for this problem as well, then we do not have to iterate over all $Z \subseteq \mathcal{U}_k$ anymore and thus greatly reduce the complexity. The only difficulty is that we can not easily adapt constraints (2), which prevent any subtours, without making the formulation non-linear. We can however consider a relaxation of this formulation, i.e. the formulation without constraints (2), and adapt it to incorporate the selection decisions. We will call this approach the *Approximate Copycat* algorithm.

In each stage k, instead of constructing an undirected graph, we will consider the directed graph $G^k = (V^k, A^k)$, where $V^k = \{l_1, \ldots, l_k\} \cup \{o\}$ are again the vertices and A^k are the arcs between all customer locations plus the origin o. These arcs are defined for any customers i and j with $i \neq j$, and we denote the arc from the location of customer i to the location of customer j as (i, j). Furthermore, similar to the formulation for Copycat in the JR problem, we define z_j as a binary variable that equals 0 if customer j is selected, and equals 1 otherwise. We then have to solve the following integer program for each stage k:

$$
\begin{aligned}
\text{minimize} \quad & \sum_{(i,j)\in A^k} c_{ij}x_{ij} \quad + \quad \sum_{j\in V^k\setminus\{o\}} r_j z_j & & \\
\text{subject to} \quad & \sum_{j\in V^k:(i,j)\in A^k} x_{ij} = 1 - z_i & i \in V^k \setminus \{o\}; \qquad & (1)\\
& \sum_{i\in V^k:(i,j)\in A^k} x_{ij} = 1 - z_j & j \in V^k \setminus \{o\}; \qquad & (2)\\
& \sum_{j\in V^k:(o,j)\in A^k} x_{ij} = 1 & & (3)\\
& \sum_{i\in V^k:(i,o)\in A^k} x_{ij} = 1 & & (4)\\
& x_{ij} \in \{0,1\} & (i,j) \in A^k; & \\
& z_j \in \{0,1\} & j \in V^k \setminus \{o\}. &
\end{aligned}
$$

Constraints (1) and (2) make sure that all selected customers are included in a tour and visited exactly once. Constraints (3) and (4) ensure that the origin is always selected. Because of this, a solution that rejects all customers can not be feasible with these constraints. That is why we should also calculate the cost of rejecting all arrived customers, and compare that to the optimal objective value from the integer program. The selection decision then depends on which value is lower.

6.2 Production Phase

For the Copycat algorithm, the production phase is very simple. Due to the fact that we have to compute $\phi(Z)$ for all $Z \subseteq \mathcal{U}$ in the selection phase, we simply retrieve the value of $\phi(\mathcal{A})$ to obtain the optimal production costs. For the Approximate Copycat algorithm, we can also compute $\phi(\mathcal{A})$ to solve the production phase. If an exact solution is not needed, another possibility is to use the 1.5-approximation algorithm from Christofides (1976).

7 Computational Results

7.1 Joint Replenishment Problem

To test the performance of the two online algorithms on the JR problem with online customer selection, we perform computational experiments. For these experiments, we require data of the demand quantities, item types and due dates of the customers. These data can be generated by sampling from a distribution. Elmachtoub and Levi (2016) make use of a uniform distribution for this. Next to that, they also look at three different scenarios, where each scenario has slightly different parameters. We will also consider the same scenarios and verify the results.

In the first scenario, the *conservative* scenario, the demands of each customer are all equal to one unit. The second scenario, which we call the *more demands* scenario, has demands that are uniformly distributed from 1 to 10. At last, the third scenario is called the *large orders first* scenario. In this scenario, the first two orders are very large, with a demand of 100 units of type 1, and are placed in periods 1 and 15. This is to model the situation where there is at least one routine customer, and because of the large demand, these two orders will always be accepted in the selection phase.

The remaining parameters for each scenario are $T = 30$, $N = 300$, $M = 3$, $K_0 = 100$, $K_1 = K_2 = K_3 = 20$, $r = 10$ and $h = 1$. All due dates and item types are chosen uniformly at random. We perform 100 experiments for each scenario, where each experiment has a different customer sequence. After every customer arrival k, we calculate the performance ratio $C(\mathcal{U}_k)/C^*(\mathcal{U}_k)$. In the end, the ratios are averaged for each stage. The results of this can be seen in Figures 1, 2 and 3. Note that in these figures, StablePair(2) means the StablePair algorithm with a constant scaling factor of 2 for the rejection costs.

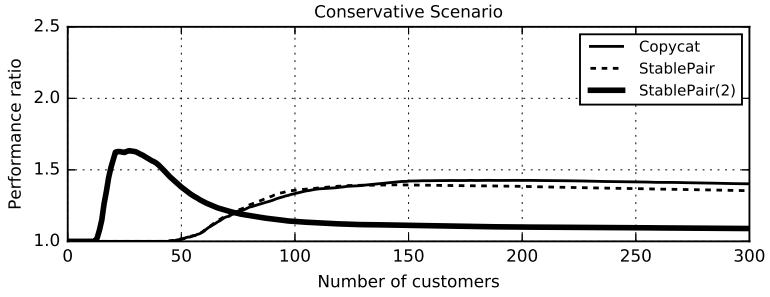


Figure 1: Results conservative scenario for JR problem.

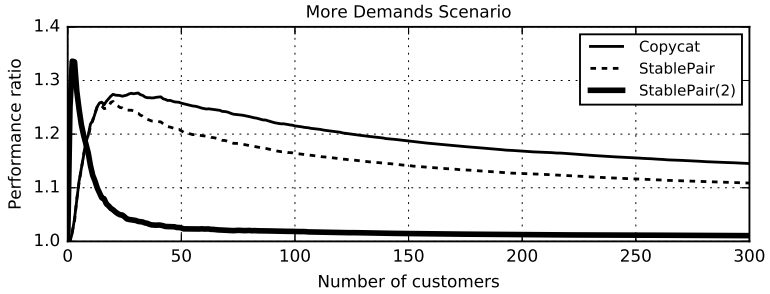


Figure 2: Results more demands scenario for JR problem.

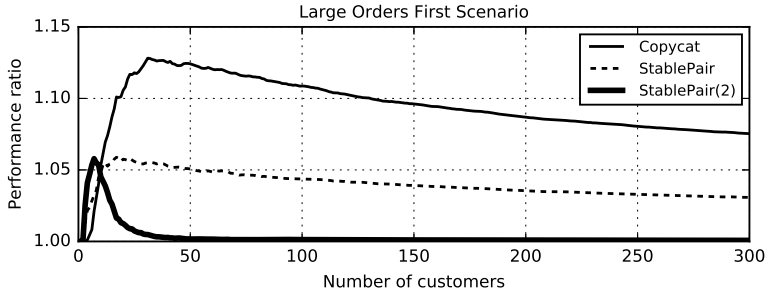


Figure 3: Results large orders first scenario for JR problem.

Comparing these figures to the ones in Elmachtoub and Levi (2016), we see that the results are similar. For StablePair(2), we see that the peaks in the beginning are a bit lower for the conservative and large orders first scenarios. Also, there is a bit of a difference in the first 100 stages for StablePair in the large orders first scenario. In the appendix, we also show the confidence intervals in each stage for the more demands and large orders first scenarios. There is also a clear difference in running time between Copycat and StablePair, where StablePair is around 80 times faster on average.

We can also look at some more interesting statistics, which are the *maximum performance ratio* and the *final performance ratio*. These are shown in Table 1 for each scenario and algorithm. The maximum performance ratio is obtained by taking the maximum ratio over all stages in the 100 experiments. This should always be lower than the competitive ratio, which was 4 for Copycat and 3 for StablePair. The final performance ratio is the mean performance ratio after the last customer arrives.

Table 1: Statistics for JR problem, with in parentheses the corresponding maximum and final performance ratios from the experiments of Elmachtoub and Levi (2016).

	Maximum performance ratio			Final performance ratio		
Scenario	Copycat	StablePair	StablePair(2)	Copycat	StablePair	StablePair(2)
Conservative	1.51 *(1.54)*	1.49 *(1.57)*	2.01 *(2.71)*	1.40 *(1.40)*	1.35 *(1.36)*	1.09 *(1.09)*
More Demands	1.71 *(1.75)*	1.64 *(1.75)*	2.83 *(3.00)*	1.15 *(1.15)*	1.11 *(1.11)*	1.01 *(1.01)*
Large Orders First	1.29 *(1.34)*	1.25 *(1.44)*	1.19 *(1.55)*	1.08 *(1.07)*	1.03 *(1.02)*	1.00 *(1.00)*

We can compare these results as well. By doing this, we can see that the final performance ratios are almost all the same as in Elmachtoub and Levi (2016), with the largest difference being 0.01. For the maximum performance ratio, we see that there are quite some differences, especially for the large orders first scenario.

For all of the results above, the 1.8-approximation algorithm was used to solve the production phase. Table 2 shows the maximum and mean optimality gap in the production

phase for each scenario, which we define as the total production cost after the rounding step divided by the total production cost from the LP relaxation. These optimality gaps can be directly compared to the upper bound of 1.8. As can be seen in the table, the maximum and mean optimality gap are both very close to 1 in all scenarios. This means that the use of the approximation algorithm does not have a big impact. Because of this, and the fact that the running time is shorter as well, the approximation algorithm is preferred over solving the production phase exactly.

Table 2: Optimality gaps for approximation algorithm in production phase of JR problem.

Scenario	Maximum	Mean
Conservative	1.0039	1.0000
More Demands	1.0089	1.0000
Large Orders First	1.0072	1.0000

7.1.1 Scaling Rejection Costs

As mentioned in section 4.4, we will look at both constant and linear scaling factors. Before we present the results of the Nelder-Mead algorithm, it might be interesting to first examine the effect of a scaling factor on the performance ratios. Figure 4 shows the results for six different constant scaling factors in the conservative scenario.

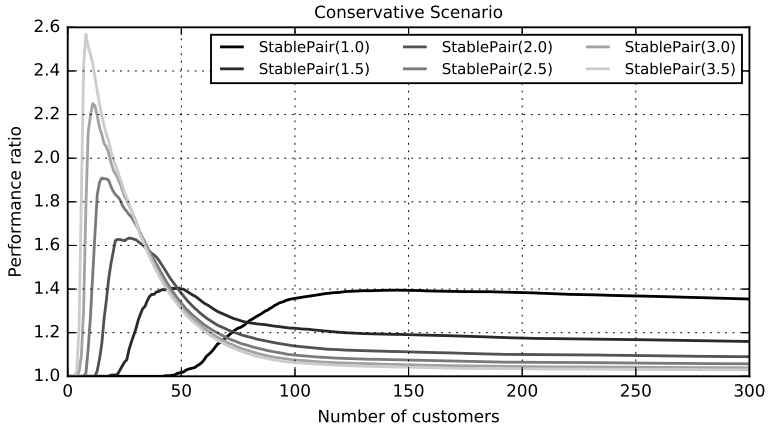


Figure 4: Results various scaling factors in conservative scenario JR problem.

It is clear that in this case, a higher scaling factor has a lower final performance ratio at the cost of a higher peak, and therefore worse performance, in the beginning. We get a similar result for the other two scenarios, as can be seen in Figures 18 and 19 in the appendix.

We therefore have a trade-off between having a low final performance ratio and a having a low maximum performance ratio. That is why we are interested in minimizing the mean performance ratio over all stages and customer sequences, where we try to get both the final and maximum performance ratio as low as possible.

Constant Scaling Factor

We apply the Nelder-Mead algorithm for the constant scaling factors of the form $\phi_k = \alpha$, with the initial points $x_1 = 1.0$, $x_2 = 4.0$ and $x_3 = 2.5$. These points were decided on after trying other initial points, which gave no improvements. The results can be found

in Table 3. We use the same 100 different customer sequences as in the previous section as input data. For all three scenarios, the three points in the final solution give the same mean performance ratio over these 100 customer sequences, which means we can choose from multiple solutions. In the table, we selected the point with the lowest label, i.e., x_1.

Table 3: Results for constant scaling factor $\phi_k = \alpha$, with $\phi_k = 2$ as comparison.

	Coefficient	Mean perf. ratio	Coefficient	Mean perf. ratio
Scenario	α	$f(x)$	α	$f(x)$
Conservative	2.4964	1.1611	2.0000	1.1679
More Demands	3.4844	1.0190	2.0000	1.0264
Large Orders First	4.0000	1.0027	2.0000	1.0033

For the conservative and more demands scenario, the other two solution points contain nearly identical coefficients. The coefficients from the three points in the large orders first scenario vary a bit more. This might have to do with the fact that there is less difference between scaling factors in this scenario, which we can see in Figure 19 in the appendix. The performance ratio lines for any scaling factor above 2.5 are almost the same. In all scenarios, the solution has a better mean performance than StablePair(2).

Linear Scaling Factor

We also apply the Nelder-Mead algorithm for the linear scaling factors of the form $\phi_k = \alpha + \beta k$, with the initial points $\mathbf{x}_1 = \begin{bmatrix} 4.0 & -0.1 \end{bmatrix}$, $\mathbf{x}_2 = \begin{bmatrix} 9.0 & -0.001 \end{bmatrix}$ and $\mathbf{x}_3 = \begin{bmatrix} 18.5 & 0.05 \end{bmatrix}$. Other initial points were tried as well, and they gave no improvements. The results can be found in Table 4, where we again used the same 100 different customer sequences as in the previous section as input data. Just as for the constant scaling factor, the three points in the final solution for all scenarios give the same mean performance ratio over these 100 customer sequences, so the solution shown is the point with the lowest label.

Table 4: Results for linear scaling factor $\phi_k = \alpha + \beta k$, with $\phi_k = 2$ as comparison.

	Coefficients		Mean perf. ratio	Coefficients		Mean perf. ratio
Scenario	α	β	$f(\mathbf{x})$	α	β	$f(\mathbf{x})$
Conservative	2.2535	0.0173	1.1607	2.0000	0.0000	1.1679
More Demands	3.5144	-0.0037	1.0190	2.0000	0.0000	1.0264
Large Orders First	18.5000	0.0500	1.0027	2.0000	0.0000	1.0033

We have the same conclusion as for the constant scaling factor, where for the conservative and more demands scenario, the other two solution points contain nearly identical coefficients. Comparing the results for the linear scaling factor with the ones for the constant scaling factor, we see that only in the conservative scenario there is an improvement. For the other scenarios, the mean performance ratio is exactly the same. Just as for the constant scaling factors, this solution also has a better mean performance than StablePair(2) in all scenarios.

7.1.2 Including Soft Capacities

Because the JR problem with soft capacities takes considerably more time to solve, we will make the customer sequences a bit smaller by only looking at 100 customers. Next

to that, we choose the following capacities: $L_{1s} = L_{2s} = L_{3s} = 25$ and $L_{0s} = 50$. We will again look at the conservative, more demands and large order first scenarios, with the other parameters remaining the same. The maximum and final performance ratios can be found in Table 5.

Table 5: Results Copycat algorithm for JR problem with soft capacities.

Scenario	Max perf. ratio	Final perf. ratio
Conservative	1.45	1.34
More Demands	1.80	1.15
Large Orders First	1.29	1.12

For the conservative scenario, there is not much of a difference compared to the problem without soft capacities. This has to do with the fact that the demands are all low, so that the capacity will not be exceeded in most cases when we only have 100 customers. Comparing the more demands scenario, we see a much larger difference. The peak in the beginning is a bit higher, but after that the performance ratios are quite a lot lower than the ones for the problem without soft capacities. The large orders first scenario is only slightly different.

7.1.3 Special Case: Economic Lot Sizing Problem

The *economic lot sizing problem* is a special case of the joint replenishment problem, where there is only one type of item. This means we can solve this problem by using the same implementation. We will again consider the conservative, more demands and large orders first scenarios. The parameters remain the same as for the JR problem, except for $M = 1$ and $r = 5$. Elmachtoub and Levi (2016) also set $N = 500$ to compute the final and maximum performance ratios, while only showing the first 300 customers in the performance ratio figures. We will do the same, and the results can be seen in Figures 5, 6 and 7.

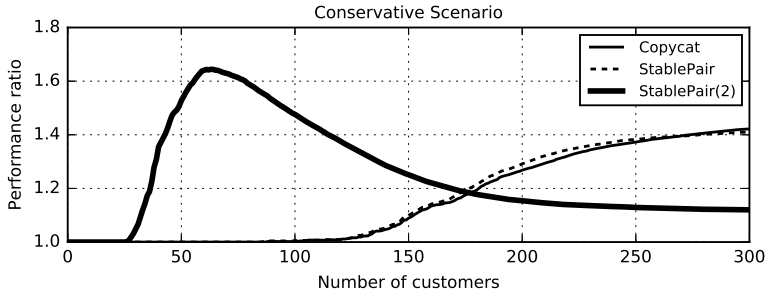


Figure 5: Results conservative scenario for ELS problem.

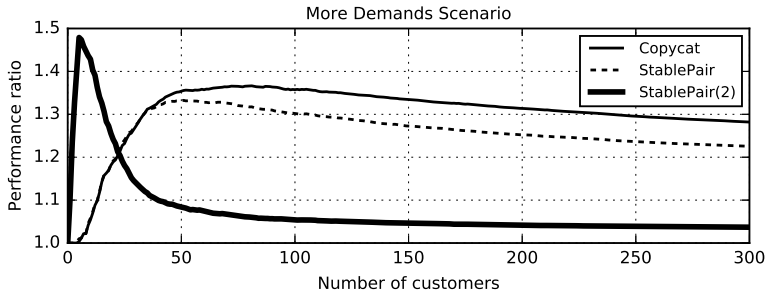


Figure 6: Results more demands scenario for ELS problem.

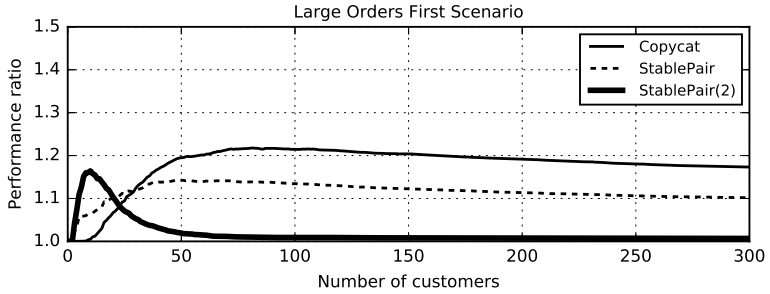


Figure 7: Results large orders first scenario for ELS problem.

We again compare these results with the ones in Elmachtoub and Levi (2016). For the conservative and more demands scenarios, the figures look identical. Only for the large orders first scenario, the three curves are a bit different. This can be seen as well when we look at the maximum and final performance ratios, which can be found in Table 6. In the appendix, we also show the confidence intervals in each stage for the more demands and large orders first scenarios.

Table 6: Statistics for ELS problem, with in parentheses the corresponding maximum and final performance ratios from the experiments of Elmachtoub and Levi (2016).

	Maximum performance ratio			Final performance ratio		
Scenario	Copycat	StablePair	StablePair(2)	Copycat	StablePair	StablePair(2)
Conservative	1.51 *(1.51)*	1.48 *(1.47)*	1.87 *(1.85)*	1.46 *(1.44)*	1.41 *(1.41)*	1.11 *(1.11)*
More Demands	1.58 *(1.54)*	1.52 *(1.54)*	2.63 *(2.40)*	1.25 *(1.23)*	1.20 *(1.19)*	1.03 *(1.03)*
Large Orders First	1.33 *(1.26)*	1.30 *(1.24)*	1.38 *(1.22)*	1.15 *(1.11)*	1.09 *(1.07)*	1.01 *(1.00)*

Similar to the statistics for the JR problem, the maximum performance ratios are again quite a bit different compared to the ones in Elmachtoub and Levi (2016), especially for the large orders first scenario. The final performance ratios for the conservative and more demands scenarios are nearly identical again, only the ratio for Copycat is 0.02 higher. For the large orders first scenario, the final performance ratios differ a bit more.

7.2 Traveling Salesman Problem

To test the performance of the Copycat and Approximate Copycat algorithms on the TS problem with online customer selection, we will perform computational experiments with three similar scenarios as in the JR problem.

In the conservative scenario, the demands of each customer are again all equal to one unit and the rejection cost per unit is equal to 10, i.e., $r = 10$. For the more demands scenario, the demands are uniformly distributed from 1 to 10 and $r = 1$. Finally, in the large orders first scenario, the first two orders have a demand of 100 units, such that they will always be accepted. We set $r = 1$ in this scenario as well.

For all three scenarios, the metric space is $\mathcal{M} = [-5, 5] \times [-5, 5]$, such that $\mathcal{M} \subset \mathbb{R}^2$. As distance measure, we use the Euclidean distance. The total number of customers $N = 10$, and the locations of these customers are uniformly drawn over the whole metric space. The origin is at $(0, 0)$. At last, the cost per unit of distance p is equal to 1 in all three scenarios. We perform 1000 experiments per scenario and then calculate the performance ratios per stage, just as before. The results of this can be seen in Figures 8, 9 and 10.

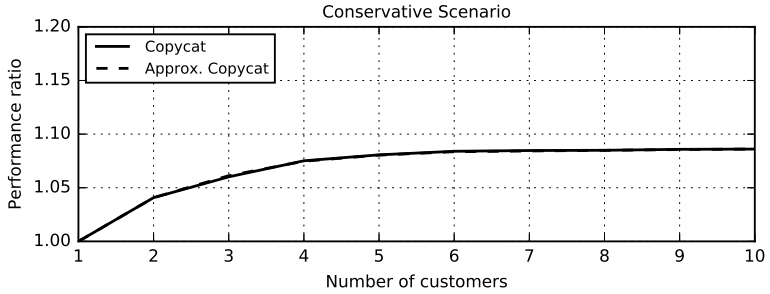


Figure 8: Results conservative scenario for TS problem.

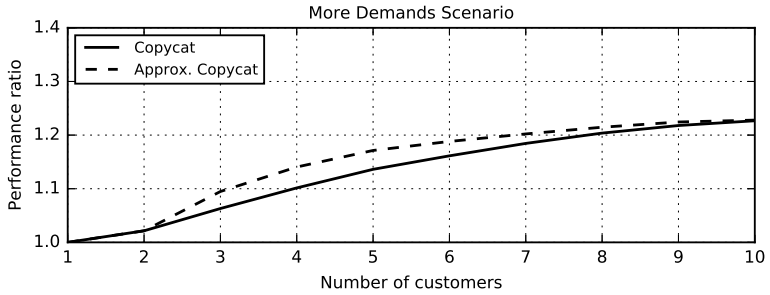


Figure 9: Results more demands scenario for TS problem.

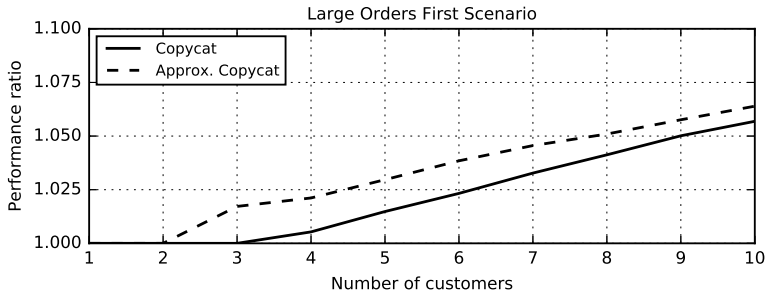


Figure 10: Results large orders first scenario for TS problem.

In the conservative scenario, we see that there is almost no difference between Copycat and Approximate Copycat. For the more demands scenario, the difference is a bit larger, but both lines come back together in the end. This can probably be explained by the fact that Approximate Copycat accepts more customers than Copycat. This has the same effect as a scaling factor, where the performance in the end will be better at the cost of a bit worse performance in the beginning stages. Sadly, we can not compute the performance ratios for any more customers in reasonable time, so we can not see if Approximate Copycat will outperform Copycat in later stages. For 10 customers and 1000 experiments per scenario, it took a bit more than 3 hours to run all three scenarios, and the running time increases exponentially for every extra customer. Finally, in the large orders first scenario, the performance ratios of Approximate Copycat are higher than Copycat after stage 2, but as can be seen in the figure, the absolute differences are very small.

Just as for the JR problem, we also computed the maximum and final performance ratios. These ratios can be found in Table 7. In the appendix, we also show the confidence intervals in each stage for all three scenarios. These figures show that even though we perform 1000 experiments, there is still some uncertainty.

Table 7: Statistics for TS problem.

	Maximum performance ratio		Final performance ratio	
Scenario	Copycat	Approx. Copycat	Copycat	Approx. Copycat
Conservative	1.88	1.88	1.09	1.09
More Demands	1.78	2.44	1.23	1.23
Large Orders First	1.43	1.51	1.06	1.06

In all scenarios, the final performance ratios of both algorithms are very close. Even though the maximum performance for Approximate Copycat is higher in all cases, the running time is much lower. On average, Approximate Copycat is about 40 times faster than Copycat to solve one experiment of 10 customers.

8 Conclusion

In this thesis, we presented a general model that can be used to solve problems with online customer selection, and we explained how we can split these kind of problems into two phases: the selection phase and the production phase. We also gave two online algorithms that can be used to solve the selection phase. As a way to show how to apply this general model to concrete problems with online customer selection, we solved the JR and ELS problems using the Copycat and StablePair algorithms. We also solved the TS problem, using the Copycat algorithm and a slightly modified version that has a lower complexity.

The results for these JR and ELS problems were compared to the results of Elmachtoub and Levi (2016), and we saw that even though most of the results are close, there were also a few differences. Especially in the large orders first scenario, which was a bit different for both the JR and the ELS problems. We also looked at the possibility of scaling the rejection costs, and we considered both constant and linear scaling factors. An interesting direction for future research would be to look at other scaling factors, e.g. non-linear scaling factors, and check if the performance can be improved even further. As an extension to the JR problem, we also solved the problem with soft capacities. We only adapted the Copycat algorithm for this problem, which takes a long time to solve the problem. It might therefore be worthwhile to determine whether the StablePair algorithm can be adapted as well for the JR problem with soft capacities. Furthermore, we derived competitive ratios for a version of the problem with soft capacities.

For the TS problem, we saw that the Copycat algorithm performed quite well. However, because of the complexity, we could only solve the problem for a few customers. The Approximate Copycat algorithm reduced a part of the complexity, and is able to solve the selection phase a lot faster. However, because we require the optimal offline costs to compute performance ratios, we were still not able to determine the performance of the two algorithms when there are more customers. For this reason, it would be interesting to find out if there exists a more efficient way to solve the offline problem. Finally, we have not been able to determine competitive ratios for the two algorithms for this problem. This could be done in future research as well.

References

Albers, S. (2003). Online algorithms: a survey. *Mathematical Programming*, 97(1-2):3–26.

Arkin, E., Joneja, D., and Roundy, R. (1989). Computational complexity of uncapacitated multi-echelon production planning problems. *Operations Research Letters*, 8(2):61–66.

Ascheuer, N., Krumke, S. O., and Rambau, J. (2000). Online dial-a-ride problems: Minimizing the completion time. In *STACS 2000*, pages 639–650. Springer.

Ausiello, G., Bonifaci, V., and Laura, L. (2008). The online prize-collecting traveling salesman problem. *Information Processing Letters*, 107(6):199–204.

Buchbinder, N., Kimbrel, T., Levi, R., Makarychev, K., and Sviridenko, M. (2013). Online make-to-order joint replenishment model: Primal-dual competitive algorithms. *Operations Research*, 61(4):1014–1029.

Christofides, N. (1976). Worst-case analysis of a new heuristic for the travelling salesman problem. Technical report, DTIC Document.

Elmachtoub, A. N. and Levi, R. (2016). Supply chain management with online customer selection. *Operations Research*, 64(2):458–473.

Epstein, L., Noga, J., and Woeginger, G. J. (2002). On-line scheduling of unit time jobs with rejection: minimizing the total completion time. *Operations Research Letters*, 30(6):415–420.

Fotakis, D. (2008). On the competitive ratio for online facility location. *Algorithmica*, 50(1):1–57.

Geunes, J., Levi, R., Romeijn, H. E., and Shmoys, D. B. (2011). Approximation algorithms for supply chain planning and logistics problems with market choice. *Mathematical programming*, 130(1):85–106.

Geunes, J., Romeijn, H. E., and Taaffe, K. (2006). Requirements planning with pricing and order selection flexibility. *Operations Research*, 54(2):394–401.

Jaillet, P. and Lu, X. (2014). Online traveling salesman problems with rejection options. *Networks*, 64(2):84–95.

Jaillet, P. and Wagner, M. R. (2008). Online vehicle routing problems: A survey. In *The Vehicle Routing Problem: Latest Advances and New Challenges*, pages 221–237. Springer.

Karp, R. M. (1992). On-line algorithms versus off-line algorithms: How much is it worth to know the future? In *IFIP Congress (1)*, volume 12, pages 416–429.

Keskinocak, P., Ravi, R., and Tayur, S. (2001). Scheduling and reliable lead-time quotation for orders with availability intervals and lead-time sensitive revenues. *Management Science*, 47(2):264–279.

Lagarias, J. C., Reeds, J. A., Wright, M. H., and Wright, P. E. (1998). Convergence properties of the nelder–mead simplex method in low dimensions. *SIAM Journal on optimization*, 9(1):112–147.

Laporte, G. (1992). The traveling salesman problem: An overview of exact and approximate algorithms. *European Journal of Operational Research*, 59(2):231–247.

Levi, R., Roundy, R., Shmoys, D., and Sviridenko, M. (2008). A constant approximation algorithm for the one-warehouse multiretailer problem. *Management Science*, 54(4):763–776.

Levi, R., Roundy, R. O., and Shmoys, D. B. (2006). Primal-dual algorithms for deterministic inventory problems. *Mathematics of Operations Research*, 31(2):267–284.

Nelder, J. A. and Mead, R. (1965). A simplex method for function minimization. *The computer journal*, 7(4):308–313.

Slotnick, S. A. (2011). Order acceptance and scheduling: A taxonomy and review. *European Journal of Operational Research*, 212(1):1–11.

Van den Heuvel, W. and Wagelmans, A. P. (2010). Worst-case analysis for a general class of online lot-sizing heuristics. *Operations research*, 58(1):59–67.

Wagner, H. M. and Whitin, T. M. (1958). Dynamic version of the economic lot size model. *Management science*, 5(1):89–96.

A Appendix

A.1 Proof of Lemma 1

Lemma 1

If the Copycat or StablePair algorithm is used for the JR problem with stable soft capacities, and $\Sigma_{i=1}^{M} K_i \leq K_0$, then $P(\mathcal{A}) \leq 5P(\mathcal{A}^*) + 5R(\mathcal{A} \cap \mathcal{R}^*) \leq 5C^*(\mathcal{U})$.

Before we can prove this lemma, we require an other lemma, which we will present first. This lemma is proved in Elmachtoub and Levi (2016), in which it is called Lemma 6.

Lemma A

Let $(Q, T) \subseteq (\mathcal{Q}, \mathcal{U})$ be a stable pair. Each setup interval, which is defined as the range of due dates that an order serves, induced by the solution to $P(Q, T)$ must have a length of at most r/h periods.

We are now ready to prove Lemma 1. Most of the proof is based on the proofs of Lemma 10 and Theorem 7 in Elmachtoub and Levi (2016), and we will use similar notation.

Proof:

We will first introduce some notation. Let $s_1, \ldots, s_m$ denote the times of the orders in the optimal production plan with soft capacities for $P(\mathcal{A}^*)$. We can decompose the costs for this optimal production plan into the setup costs K^* and the holding costs H^*. Let $\mathcal{A}^i$ denote the type i customers in $\mathcal{A}$. For each $k \in \mathcal{A}^i$, let a_k be the order date that serves customer k in the stable pair that the StablePair algorithm used to accept the customer. We place all a_k in a set $\hat{T}^i$, and sort them from earliest to latest. Next, construct $T^i \subseteq \hat{T}^i$ by removing any order date that is within r/h periods of the previous one. Furthermore, we denote X^i as the set of all type i customers that have a due date in the interval $[t, t + r/h]$ for some $t \in T^i$. Finally, we define the set $Y^i = \mathcal{A}^i \setminus X^i$.

The rest of the proof is as follows. We will construct a solution that serves all accepted customers, and use it to find an upper bound on the total production cost. This will be done in two steps. We will first find an upper bound on the total holding costs. This part directly comes from the proof of Lemma 10 in Elmachtoub and Levi (2016), but we will show it here for completeness. Secondly, we will bound the total setup costs. This part is new, and extends the proof of Theorem 7 in Elmachtoub and Levi (2016) to work for multiple item types.

We first decompose the set of accepted customers $\mathcal{A}$ into $\mathcal{A} \cap \mathcal{A}^*$ and $\mathcal{A} \cap \mathcal{R}^*$. We can easily serve all the customers in $\mathcal{A} \cap \mathcal{A}^*$ by using the same orders as in the optimal production plan for $P(\mathcal{A}^*)$, i.e., the orders $s_1, \ldots, s_m$. By doing this, we incur a production cost of $K^* + H^*$.

To serve the customers in $\mathcal{A} \cap \mathcal{R}^*$, we first duplicate the orders $s_1, \ldots, s_m$, and shift them back by r/h periods, such that the new orders are at $s_1 - r/h, \ldots, s_m - r/h$. This adds another K^* to the production cost. Next, we will consider each item type i separately. Assume for now that for each $t \in T^i$, there exists an order s_j in the production plan $P(\mathcal{A})$ such that either $s_j \in [t - r/h, t]$ or $s_j - r/h \in [t - r/h, t]$. Furthermore, this order satisfies the capacity constraints, i.e., enough orders are placed on that order date such that all demand is satisfied. By assuming this property holds, we can derive an upper bound on the holding costs for all customers in $\mathcal{A} \cap \mathcal{R}^*$. Specifically, the customers with a due date in $[t, t + r/h]$, which by definition are the customers in $X^i \cap \mathcal{A}^i \cap \mathcal{R}^*$, can be served with total holding costs of at most $2R(X^i \cap \mathcal{A}^i \cap \mathcal{R}^*)$. The remaining customers

left to be served are in $Y^i \cap \mathcal{R}^*$. For any customer $k \in Y^i \cap \mathcal{R}^*$, there exists a $t \in T^i$ such that $t \leq a_k \leq t + r/h \leq a_k + r/h$. Next to that, it follows from the definition of a_k and Lemma A, that $t_k \in [a_k, a_k + r/h]$. This means that the holding cost from the order to t, t to a_k, and a_k to t_k, is at most $3r$ per unit. This means the total holding cost for the customers in $Y^i \cap \mathcal{R}^*$ will be at most $3R(Y^i \cap \mathcal{R}^*)$.

The only thing left to do is ensure that the property holds by placing extra orders where needed. We will first look at the orders for each type i. For every $t \in T^i$, if $s_j - r/h \leq t < s_j$, we will place extra orders for item i at the order date $s_j - r/h$. Otherwise, we will place these orders at the order date s_j. For every order date j, let $\mathcal{A}^i_j \subseteq \mathcal{A}^i$ be the set of accepted type i customers that are served by an order on this date. We will only consider the customers that still have to be served, i.e., the customers in $\mathcal{A}^i_j \cap \mathcal{R}^*$. Let $O_i L_i + o$ be the total demand of these customers, where O_i and o are both integers and $o < L_i$. This means we require $O_i + 1$ extra orders to serve the customers. We can bound the setup costs for these orders by noting that $rL_i \geq K_i$, otherwise no customers of type i would be accepted. Therefore, the setup costs of these orders are:

$$(O_i + 1)K_i = O_i K_i + K_i \leq rO_i L_i + K_i \leq R(\mathcal{A}^i_j \cap \mathcal{R}^*) + K_i.$$

By summing these costs for all types and using that $\Sigma_{i=1}^M K_i \leq K_0$, we get that the setup cost of all types per order date is at most $R(\mathcal{A}_j \cap \mathcal{R}^*) + K_0$. We can then sum this for all order dates, and we get that the setup cost for all extra orders of all types is at most $R(\mathcal{A} \cap \mathcal{R}^*) + 2K^*$. For the joint orders, we can use a similar analysis, and we obtain that the setup costs will be at most $R(\mathcal{A} \cap \mathcal{R}^*) + K^*$.

Combining all these costs, we get the following:

$$P(\mathcal{A}) \leq 5K^* + H^* + 2R(\mathcal{A} \cap \mathcal{R}^*) + \Sigma_{i=1}^M (2R(X^i \cap \mathcal{A}^i \cap \mathcal{R}^*) + 3R(Y^i \cap \mathcal{R}^*)).$$

This can be reduced to:

$$\begin{aligned} P(\mathcal{A}) &\leq 5K^* + H^* + 5R(\mathcal{A} \cap \mathcal{R}^*) \\ &\leq 5P(\mathcal{A}^*) + 5R(\mathcal{A} \cap \mathcal{R}^*) \\ &\leq 5C^*(\mathcal{U}). \end{aligned}$$

■

A.2 Confidence Intervals

For the conservative scenario in the JR and ELS problems, there is less uncertainty in the demands. This means the confidence intervals are small and follow the performance ratio line almost perfectly. This does not give any interesting figures, therefore we will not show them here.

A.2.1 Joint Replenishment Problem

Figure 11: 95% confidence intervals for more demands scenario in JR problem.

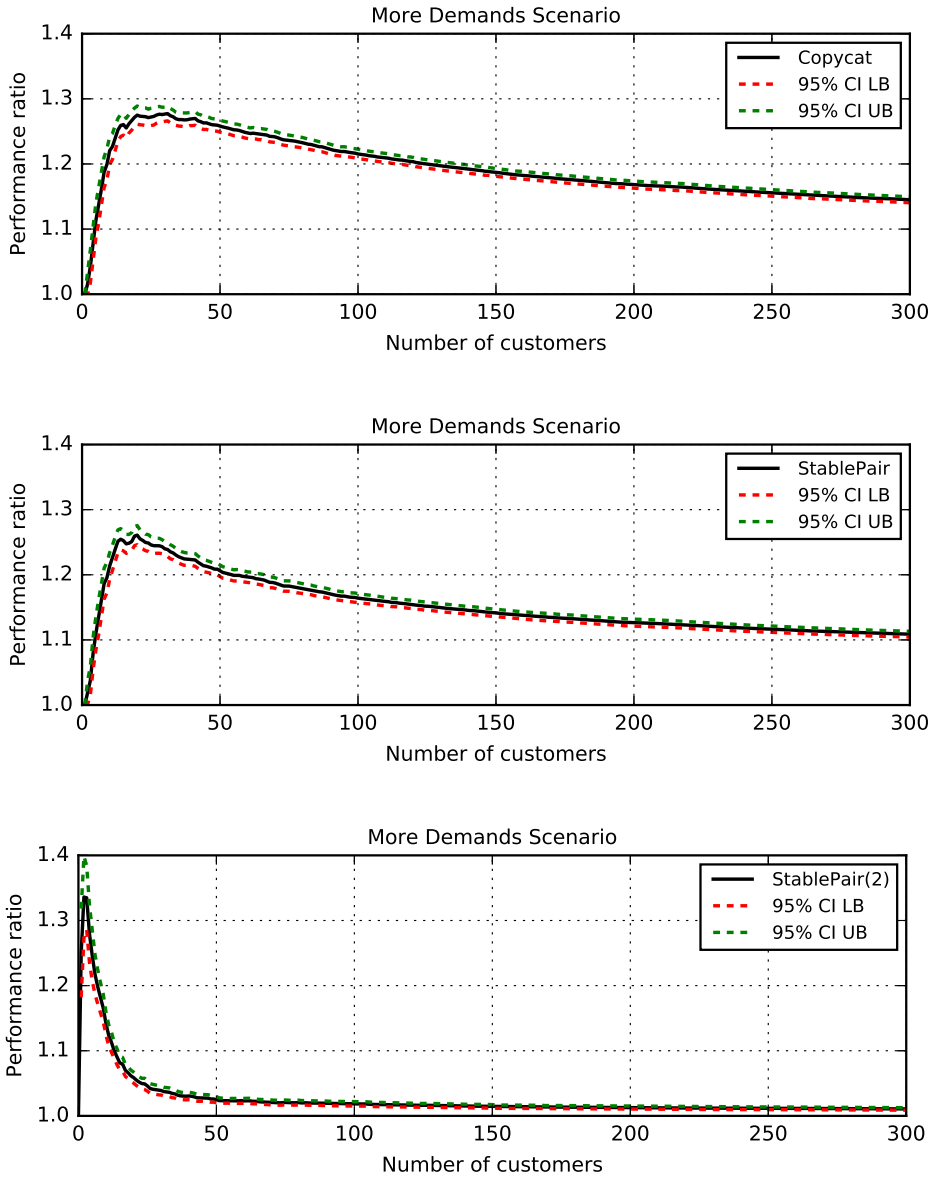
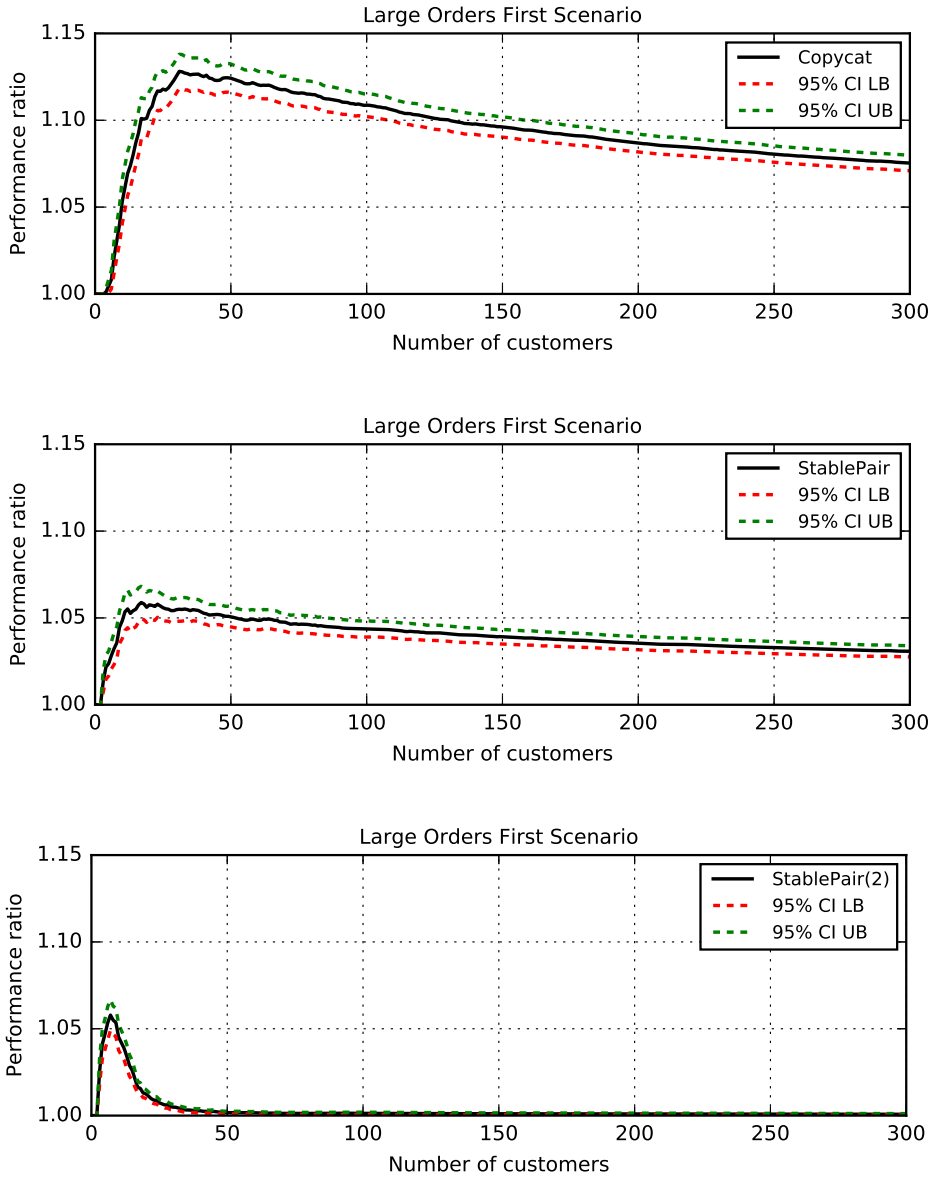


Figure 12: 95% confidence intervals for large orders first scenario in JR problem.



A.2.2 Economic Lot Sizing Problem

Figure 13: 95% confidence intervals for more demands scenario in ELS problem.

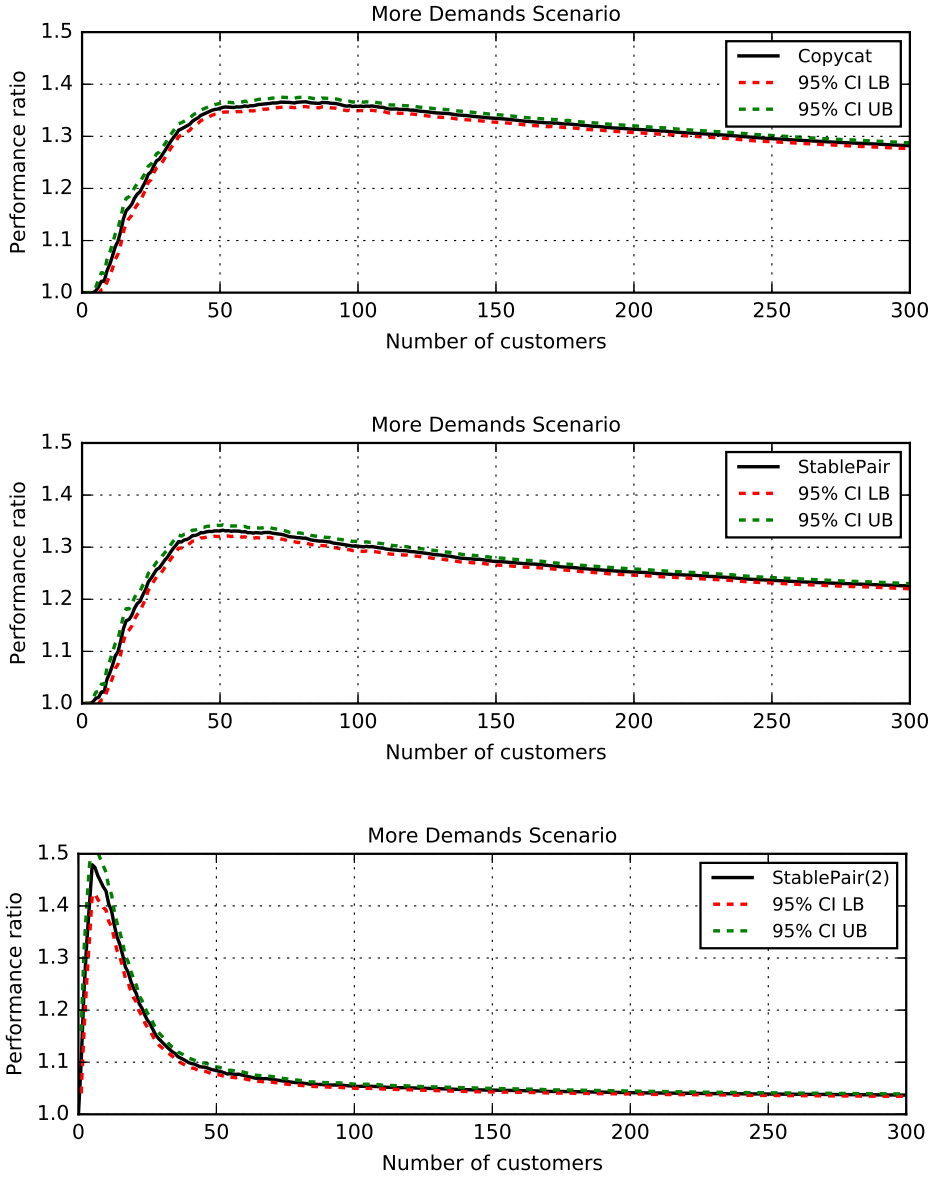
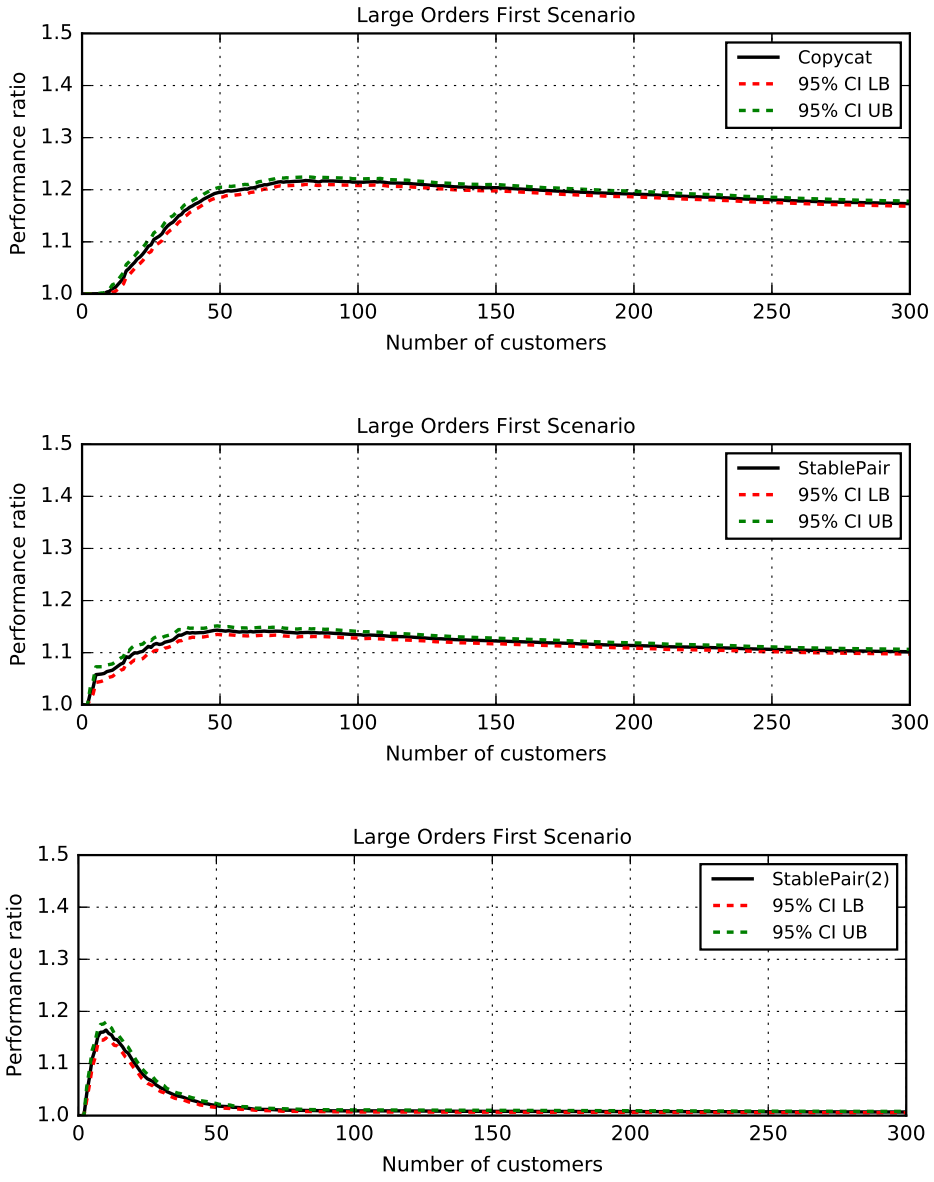


Figure 14: 95% confidence intervals for large orders first scenario in ELS problem.



A.2.3 Traveling Salesman Problem

Figure 15: 95% confidence intervals for conservative scenario in TS problem.

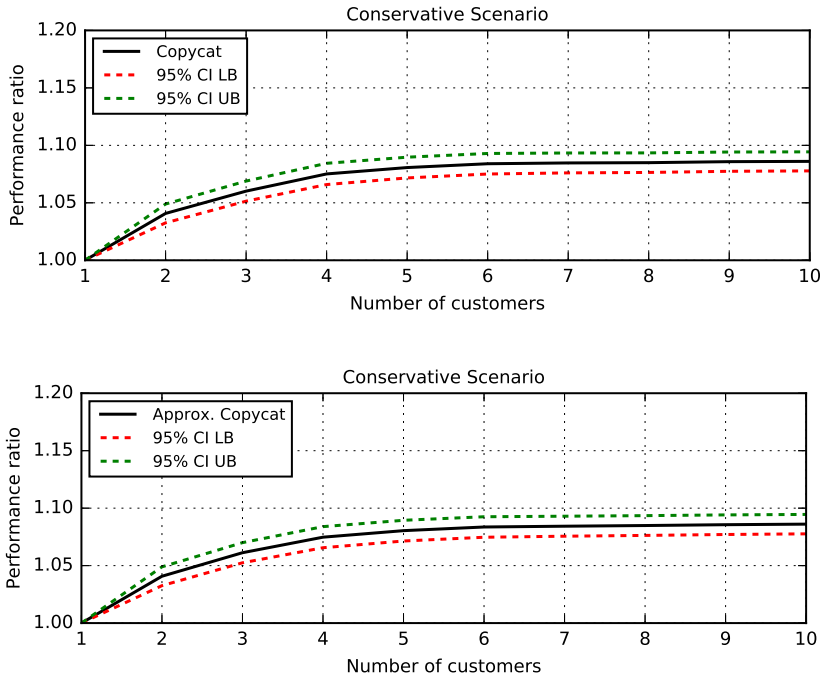


Figure 16: 95% confidence intervals for more demands scenario in TS problem.

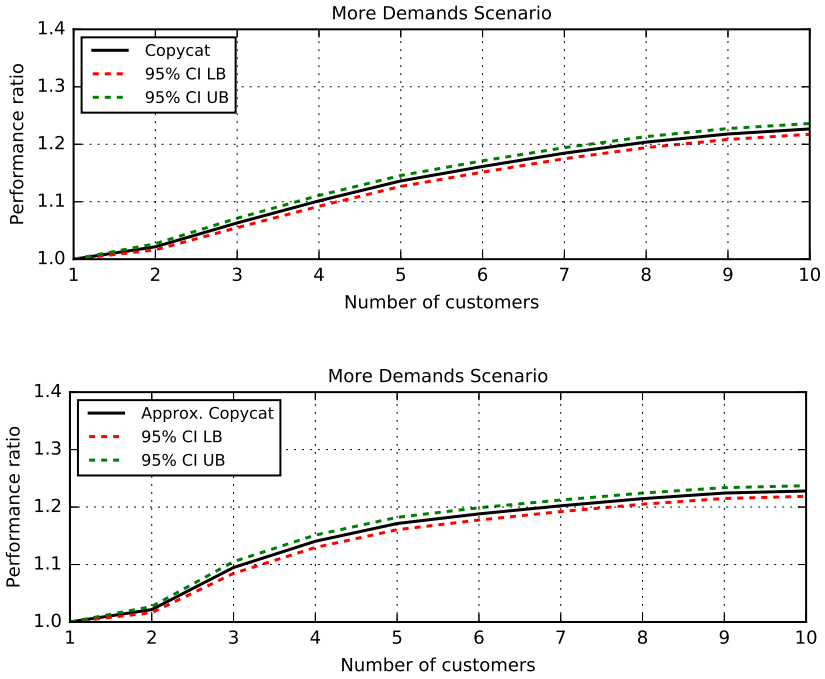
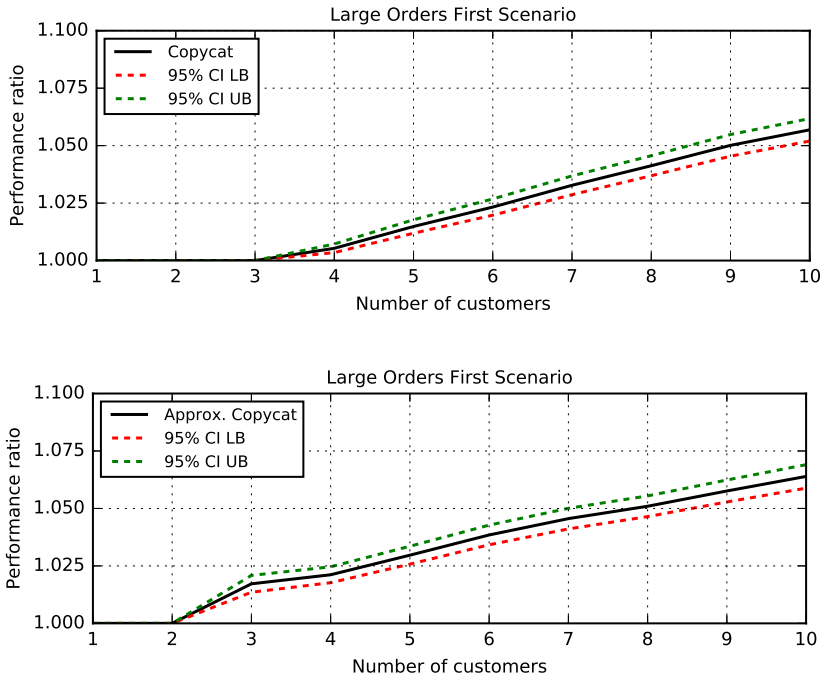


Figure 17: 95% confidence intervals for large orders first scenario in TS problem.



A.3 Scaling Factors

Note that the theoretical competitive ratios that were derived are only for the Copycat and *unscaled* StablePair algorithms. This means that the performance ratios obtained when using a scaling factor can exceed the competitive ratio.

Table 8: Results various scaling factors in JR problem (100 experiments, 300 customers).

	Maximum performance ratio						Final performance ratio					
Scenario	1.0	1.5	2.0	2.5	3.0	3.5	1.0	1.5	2.0	2.5	3.0	3.5
Conservative	1.49	1.61	2.01	2.47	3.20	3.75	1.35	1.16	1.09	1.06	1.04	1.03
More Demands	1.64	1.90	2.83	3.00	3.00	3.25	1.11	1.03	1.01	1.00	1.00	1.00
Large Orders First	1.25	1.19	1.19	1.24	1.24	1.24	1.03	1.00	1.00	1.00	1.00	1.00

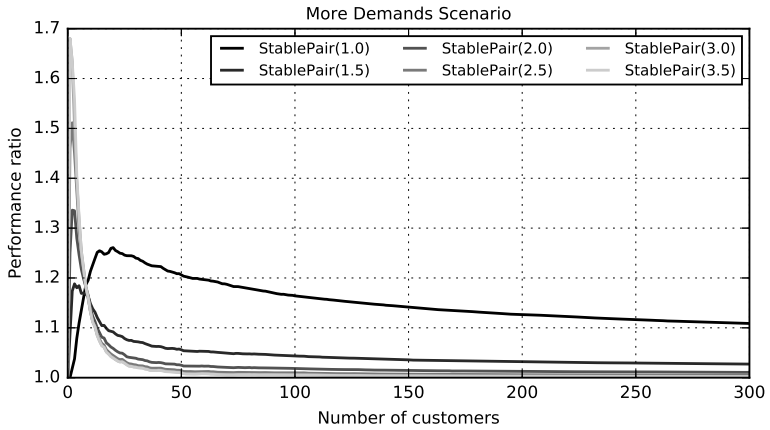


Figure 18: Results various scaling factors in more demands scenario JR problem.

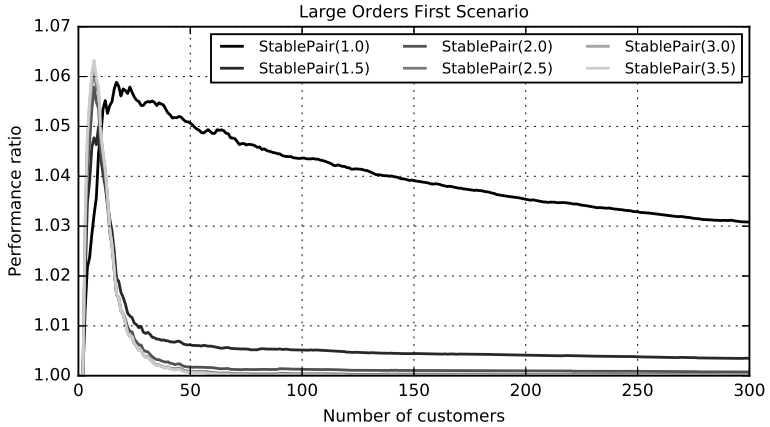


Figure 19: Results various scaling factors in large orders first scenario JR problem.