

Optimization of Restaurant Customer Seating

A Decision Based Approach

Abstract

Scheduling problems are a broad class of challenges that require optimal allocation of resources to accomplish a specific task. The process of seating customers at a restaurant is one such task, in which the resources are the collection of tables in the restaurant. One way to approach this problem is to consider each new customer as a decision. They request a table for their party, and the establishment may decide whether to accept or decline. The decision is based on a number of variables. The size of the party, availability of tables, expected duration of stay, and revenue from the bill are all considered.

In this project, a restaurant is modeled as a collection of tables, and customer flow by a random process. A static decision-based algorithm was then defined and implemented to seat customers in real time as they arrive at a restaurant. The layout of the restaurant and average customer flow were based on a Vermont restaurant, Café Provence. This decision-based approach was evaluated by comparing its performance to a more naïve algorithm in a series of Monte Carlo simulations. Analysis of these simulations shows that the decision-based approach performed significantly better than the naïve algorithm when customer demand was high and certain threshold values were chosen. The success of a decision-based algorithm in modeling seating at a restaurant shows that this approach is viable in scheduling problems, and may be a useful technique in solving other scheduling challenges.

Problem

Restaurants are perhaps one of the most difficult businesses to manage in a profitable manner; doing so requires a balance of emphasis on revenue with meeting expectations of customers and staff. The seating process embodies all of these challenges: the number and identity of the customers sat at any given time directly influence revenue; the orientation in which they are seated may impact customer satisfaction, as well as the ability of the staff to do their jobs. The complex task is dynamic, as customer flow is neither fixed nor known, and considerations of table sizes, customer waiting time, distribution of customers between serving staff, and kitchen work-load are necessary. In essence, the seating of customers in a restaurant is an optimization problem. The objective is to maximize revenue for the establishment while satisfying customer needs, without putting undue strain on the staff. This problem is constrained by the size and layout of the restaurant, available staff, and kitchen capacity.

On a basic level the goal of a restaurant seating model should be to reflect the randomness and unpredictability of customer flow while maintaining some basic procedure when addressing customers. Even when demand fluctuates, restaurant staff have their own procedures which are still followed. Good models are flexible, and handle more unpredictability in both customer behavior and staff constraints in such a way that does not compromise operation standards. The best models are dynamic, considering not just the static decision, but the implications for future performance. This project will focus on the more basic goals: modeling random customer flow and addressing a simplified static decision problem.

Literature Review

The importance of the seating problem in the restaurant business has not gone unnoticed. In an article published at the Cornell School of Hotel Administration, Kimes, Barrash, and Alexander discussed the importance of revenue management in day-to-day operation [1]. The authors performed a case study on a local restaurant, Coyote Loco, gathering performance data through direct observation, in order to identify problem areas and suggest solutions. One of the key metrics they used to measure

performance was RevPASH – revenue per available seat hour [1]. They saw the goal of any revenue management strategy to be to maximize RevPASH at any given time.

Kimes, Barrash, and Alexander offered recommendations to Coyote Loco upon completion of their study for ways to improve revenue management. One of their suggestions was to improve table management in order to better predict when each table might be available for new customers [1]. They also made recommendations based on customer demand. During “cold” times, in which demand was relatively low, their strategy involved methods to raise the average bill of parties in the restaurant, such as suggestive selling and promotional sales [1]. For “hot” periods of high demand, they suggested maximizing customer volume by implementing strategies to shorten the length of time spent at a table [1]. Most of these strategies were mutually exclusive; the same suggestive selling and promotional sales that were encouraged for cold periods were avoided during hot periods in order to streamline the experience and allow more customers to be served [1].

A computational approach to the seating problem was presented by Alfio Vidotto in his 2007 PhD thesis on the use of constraint programming as applied to restaurant table management [2]. Vidotto developed a complex, dynamic model that was designed so that inexperienced users could take reservations and seat tables, while the model considered seating plans, available resources, and table usage [2]. His model was intended to operate in real time to handle unplanned events, such as late arrivals or lingering customers, and react accordingly by controlling the propagation of future delay [2]. He also took into account flexibility, for example by allowing the model to suggest alternative booking times [2].

Vidotto’s approach to solving the seating problem was to use constraint programming. A constraint program consists of a set of independent, decision variables, the set corresponding to their respective domains, and a set of constraints that acts upon the decision variables [2]. He discussed the constraint search problem – finding a solution set to the given constraints – as well as constraint optimization problems, which entail returning the best of all possible solutions [2]. Vidotto implemented a sophisticated approach to solving such problems, using multiple heuristics and time slicing [2]. The software was tested for six months at a local restaurant, Eco, and was deemed successful; it was received positively by the staff, and was found to improve the efficiency of table usage and flexibility in the seating plan [2].

A flexible decision-based approach to taking reservations was explored by Jake Feldman in his PhD thesis regarding scheduling optimization [3]. In his model, each request was considered in regards to how it would affect future seating arrangements, and the establishment had the choice to either accept, reject, or propose a counter-offer, which the customer may then consider. Feldman identifies a key decision that must be made by the restaurant if the customer’s request is not immediately accepted; they may offer a table that is oversized for the party, at the risk of losing revenue by wasting seats, or offer a table of appropriate size at a different time, which may be rejected [3]. Feldman implemented his flexible model using data from a small restaurant, compared it against three naïve seating models, and found that it outperformed all of them, improving seat utilization by between 15 and 35 percent [3].

Model

This project seeks to model of a medium-sized French restaurant in Brandon, Vermont, Café Provence. The restaurant consists of 18 tables: 13 tables for four, two tables for five, two tables for six, and one table for eight. Often on weekends and busy evenings during the summer months, the restaurant is divided into three sections of six tables, each serviced by one server. Two methods of seating were considered: a naïve, section-based seating algorithm motivated by keeping the sections

even, and a decision-based algorithm which considers the chance that other, more optimal customers may come in.

The number of tables available and their capacities are important variables in this problem. While the capacities are constant, the number available may change over time as diners pass through the restaurant. Customers are another source of variables; the probability of a party booking a reservation or walking in, and the probability they are of a given size are random variables. Each party then has several associated variables, including the number of people, time of arrival, duration of stay, and bill cost. The total restaurant revenue is another important variable, related to the sum of bills for each party over the evening.

Assumptions

Several assumptions were made in order to simplify the complex, dynamic problem of seating. First, the dynamic nature was removed from the problem. No unexpected events, including cancellations, lingering diners, late arrivals, or changed party sizes were considered. The model is also inflexible: if a reservation or walk in party requests a time that is unavailable, no counter-offer is proposed as in Feldman’s model [3]. The average check for each party was assumed to be 30 dollars per person, and every party stayed in the restaurant for exactly one and a half hours. If a customer was seated at 5:00 pm, their table would be available to be sat again at 6:30.

An average summer weekend evening was considered as the baseline for testing the seating algorithms. The average number of customers served on Friday and Saturday nights in August, the busiest month at Café Provence, for the last three years was 111 customers. Using personal experience, the probability of a given party being of a given size was determined and recorded in Table 1. It was assumed that parties arrived only in sizes from one to eight and that the probabilities for each party size are the same for reservation and walk in parties. Using a weighted average, the average party size was determined to be 3.55.

Table 1. Probabilities for Party Sizes

Party Size	1	2	3	4	5	6	7	8
Probability	0.07	0.35	0.10	0.22	0.07	0.12	0.03	0.04

There were also general assumptions made for the restaurant. If the house was divided into three sections, the tables were appropriated according to Table 2. The tables in the restaurant were assumed to not be combinable or reconfigurable to accommodate different-sized parties. Seat and table availability was considered to be the only limitation to restaurant capacity; it was assumed that adequate staff and resources were on hand to service as many people as showed up in the time allotted. Only the dinner shift was considered for this project: the first seating is at 5:00 pm, and the last is 9:00 pm. It was assumed that reservations were taken during the 7 day period before the evening, and that no more than one customer would call to reserve a table per hour. Customers were permitted to make reservations for any time at 15 minute intervals between these times. Walk in customers were accepted until 9:00.

Table 2. Number and Size of Tables in Each Section

Section	Table Capacities			
	4	5	6	8
1	4	1	1	0
2	4	1	0	1

3	5	0	1	0
Entire House	13	2	2	1

Naïve Algorithm

The naïve seating approach is a simple algorithm that distributes customers with the goal of maintaining equality in the total number of customers served – called covers – in each server’s section. Whenever a reservation is scheduled or a walk-in party requests a table, they are seated in the section with the lowest covers for the night. If a table is unavailable for the time requested in this section, the section with the next lowest covers is checked, and so on, until an available table is found and the party sat, or no available tables are found. If no tables are available the reservation or walk in party is not offered a table. Such an algorithm forms the basis of the strategy used by many restaurant hosts in seating, as part of their job is to divide customers evenly among serving staff.

The advantages of such a naïve algorithm is in its simplicity. It is easily implemented in an actual restaurant setting and in computational models. Additionally, it tends to do a good job of keeping the covers even for each section at the end of the night. This is important, especially for restaurants in which servers do not pool their tips, such as Café Provence. One of the key duties of a host or floor manager is to ensure that each server takes on the same number of people, so they have the opportunity of earning about the same amount in tips for the evening.

Such a model also comes with its disadvantages, namely in its poor table efficiency. The only restriction on seating parties at an available table is that they must not be larger than the capacity of the table. No restrictions exist on seating parties at tables with large capacity. If the only table available is much larger than the party requesting it, they will be seated, allowing those unused seats to go to waste. This phenomenon is even more likely when a section-based model is introduced; by searching in the section with fewest covers first, it is possible to seat a party at a table that is too large, while another table that would exactly accommodate the party is available in another section.

Decision-Based Algorithm

A different approach to the naïve algorithm is to decline to seat walk-in parties if they are too small for the available tables. If this was done all the time, no parties of one, two, three, or seven would ever be sat at Café Provence, because there are no tables of those exact sizes. Instead, this decision is based on the probability that at least one larger party of the appropriate size will request a table in the next hour is considered. For a party of two at a table that seats up to four, this is the chance that at least one party of three or four will enter the restaurant within the hour. Additionally, the number of available tables are considered. If many tables are available, the best decision is to seat the party, regardless of their size.

To consider both the probability of a larger party entering and the available tables, a decision quotient is defined.

$$d(s, c) = \frac{1 - (1 - (P(s+1) + P(s+2) + \dots + P(c)))^{60}}{n}$$

In the above definition, d is the decision quotient, s is the size of the incoming party, c is the table capacity of the smallest available table of appropriate size, P(x) is the probability of one party of size x walking in during any given minute, and n is the number of available tables of capacity c for the time

interval requested. If $s = c$, d is defined as 0. When the value of d is larger, this is an indication that there is a good chance of a larger party walking in, or that few tables are available.

A threshold, T , between 0 and 1, is defined along with a function f that determines whether or not to reject the party in question based on the value of d .

$$f(d, T) = \begin{cases} 0, & d < T \\ 1, & d \geq T \end{cases}$$

When $f = 0$, the party is not rejected and instead sat at the table. If $f = 1$, the likelihood that a better party will come along soon is considered high enough that it's worth declining the current party.

The advantage of such a decision based algorithm is that it utilizes tables more effectively. By declining to seat parties at tables that are too large for them, space is saved for when parties of that size come along. The model also considers that when many tables are available, it is usually best to seat parties, of any size.

One disadvantage of this method, when compared to the naïve one, is its lack of regard for the relative section totals. For the purposes of this project, the decision-based algorithm seated customers as if the entire restaurant was one section. A similar algorithm could feasibly be designed and implemented to seat using a decision approach and split the restaurant into sections.

Implementation Approach

MATLAB was used to simulate the flow of customers into a restaurant and implement the seating algorithms. Two important functions were written: `simNightNaive` and `simNightOptim`. These functions were tasked with performing a simulation of one night in the restaurant with their respective seating algorithm. The functions accepted a value for the average expected number of covers for the night, the fraction of parties which make reservations, and in the case of `simNightOptim`, a decision-making threshold. These functions returned a structure that contained several measures of performance for the night, namely the total number of covers served, average and maximum RevPASH, and the number of customers who were not offered seats.

These two functions and their respective helper functions used MATLAB's built-in random number generators to simulate customers in two stages. The first was the reservation stage, implemented by `simResNaive` and `simResOptim`. The 7 days before the night were considered to be the times in which customers could make reservations. Each day was divided up into 10 operating hours. Using the average expected covers, average party size, and fraction of reservations, the probability of a customer calling to make a reservation was calculated. For each hour during these days, a random number was generated and compared to the probability of a reservation to determine if a call was received. If so, another random number was selected and compared against the probabilities of party sizes to determine the size of the party. A third random number between one and 17 was selected for the time of the reservation; a time of one corresponded to 5:00 pm, while 17 represented a 9:00 pm reservation.

Walk in parties were handled in a similar fashion, except the process was divided up over a different interval. Each of the four hours of the shift was divided into 15 minute intervals, which were in turn divided in to one minute intervals. Random number generation was used to determine whether a party walked in, and their size. They were then assigned the time of the current 15 minute interval. For example, a party entering at 6:07 pm was considered to have entered at 6:00 pm.

A matrix was used to store the information about available tables in the restaurant at any given time. This matrix, M , had the following format: M_{ij} represents the number of available tables of capacity j at time i . For the naïve simulation, three such matrices were used to hold the information about

available tables for each section; in the decision-based simulation, only one such matrix was used. The size of such a matrix was 17 by 8, because 17 time intervals were chosen, and 8 table sizes were possible.

The task of calculating revPASH was delegated to calcRevPASH, which accepted a list of the party information from the entire night as its input. It then calculated revPASH as described by Kimes [3], by first calculating the number of available seats for any time interval. An available seat was defined to be any seat not occupied by a customer. This function considered all time intervals between 5:00 and 10:15, which is the latest a party could stay in the restaurant under the given assumptions, and calculated the number of seats filled at any given time. This quantity was subtracted from 82, the total number of seats, to yield the number of available seats at any time. The revenue for one interval was considered as the sum of the product of average check and number of seats filled, divided by the number of time intervals each party would spend at the establishment – in this case 6. RevPASH was then calculated as follows:

$$revPASH_i = \frac{S_i \alpha}{(S_0 - S_i) t}$$

In the above expression, $revPASH_i$ is the revPASH for time interval i , S_i is the number of seats occupied during interval i , α is the average check total per customer, l is the length of time in hours of one interval, t is the length of time in hours each party spends at the restaurant, and S_0 is the total number of seats in the restaurant.

A Monte Carlo simulation was utilized to compare the effectiveness of the two models over many nights. The script findBestThreshold was written to do so. Each Monte Carlo simulation ran the two algorithms for one night, saved the output data in an array, and iterated N times, using values of N between 1000 and 5000. findBestThreshold chose several possible values for the threshold, T , and then performed a Monte Carlo simulation for each, storing the averages of the data for each night in another array. This data was plotted for both the naïve and decision-based methods against the values of T in order to see how the choice of threshold affected performance.

Analysis

Algorithm Comparisons

The decision-based algorithm was evaluated relative to the naïve algorithm over many Monte Carlo simulations that varied two parameters: average nightly covers and threshold value. Threshold values from 0 to 1, in increments of 0.01, for the decision method were compared to the naïve algorithm for nightly covers of 60, 80, 100, 111, 120, 140, and 160. Plots comparing revPASH, covers, and rejected covers between the two methods may be found in the Appendix. Figure 1 corresponds to an average nightly covers value of 60, Figure 2 to 80, Figure 3 to 100, Figure 4 to 111, Figure 5 to 120, Figure 6 to 140, and Figure 7 to 160. These average cover values represent the range from a very slow weekend evening to well over the normal amount customers that Café Provence typically services in one evening. This analysis focuses on average revPASH as the primary indicator of performance.

First, the plots were examined for general trends. It is evident that for all values of average nightly covers, fewer customers were actually serviced. This is because average nightly covers was used to determine how many parties attempted to dine at the restaurant; customers who requested tables when none were available were declined, and Figures 1 through 7 show that the number of rejected customers by the two methods increases when the average covers are higher. It may be estimated that

the sum of covers and rejected covers for each trial is equal to the total average covers parameter; this is supported by approximating this value from the graph.

Next, the plots were observed to determine the optimal value of the threshold, T . When $T = 0$, no customers were offered seats. If $T = 1$, all customers were offered seats if a table was available. It was expected that the average revPASH would approach a local maximum at some value of T between 0 and 1, which would indicate the existence of an optimal value. Instead, the average revPASH increased quickly at low T , and leveled off as T was increased until it appeared to reach a state of slow, constant increase. In reality, the maximum appeared to be close to, if not at, $T = 1$; Figure 7 seems to show this is true for average covers of 160. However, Figure 6 reveals a maximum closer to $T = 0.9$. Such variation may have been due to actual trends, or a result of random variation; the choice of N may impact such fluctuations. The state of steady increase was reached more quickly for lower values of average covers. For average covers of 160 (Figure 7), this point appears to be reached by $T = 0.5$. Figure 2 shows that a steady state seems to be reached by $T = 0.3$ for an average of 80 covers per night.

Observation of the plots shows that for high load, considered to be when the average covers were greater than or equal to 120, the decision algorithm appears to outperform the naïve method when $T > 0.4$ in all categories. When 160 covers were expected, the average revPASH of the decision method at $T = 0.9$ was about 5.0, while that of the naïve method remained around 4.5, as shown in Figure 7. When expected covers was lower, the benefit of the decision-based model was less significant. Figure 4 shows that at average covers of 111, the decision-based model at $T = 0.9$ produced average revPASH of just over 3.0, while the naïve method produced a value of about 2.8. At average covers of 60, Figure 1 shows that the benefit of the decision-based model is difficult to conclude, as both methods produce an average revPASH of about 1.3.

Limitations

The many simplifying assumptions and choices made during implementation limit the effectiveness of the model and the decision-based seating algorithm. One such choice was not to apply a decision-based approach to taking reservations; this choice was made because most restaurants, including Café Provence, make it their policy to honor reservation requests on a first-come-first-served basis, because they take the initiative to call ahead. The decision method was therefore only applied the threshold to walk in customers. For a restaurant like Café Provence that serves as many or more walk-in parties as reservations, this likely had little effect, as reservations tend not to fill up the entire house. However, for a restaurant that only accepts reservations, and is consistently booked to capacity, applying a decision-based seating scheme to reservation booking might be desirable.

Another limitation of the decision-based approach was its treatment of the sections. This method considered only one section – the collection of all tables in the restaurant. The naïve method considered three. This organizational choice may have been the reason behind the comparative success of the decision-based model. In fact, when T was chosen to be equal to one, the decision-based model was better than the naïve approach, and possibly optimal. In this scenario, no decision is actually made; all parties are seated if a table with a sufficient capacity is available somewhere in the restaurant. Since the two methods differed in their choices of sections, it is difficult to conclude whether the decision making process had any affect at all.

The time taken to run the simulations may have limited the significance of the gathered data. Considerable variance is present in the performance data. Figure 5 shows the result of gathering data for 120 average covers, which was run with the highest N value of 5000 and has the least variation. Time data showed that each iteration, consisting of simulating one night with the naïve method and one with the decision-based method, took about 0.10 seconds when run on a Dell Inspiron laptop with a 2.00 GHz

Intel i7 processor. At this rate, one Monte Carlo simulation of $N = 5000$ took about 50 seconds. The `findBestThreshold` script performed 100 of these simulations at various values of T , which took about 83 minutes. In collecting the desired data for several average covers, the value of N had to be reduced to 1000 so as to obtain the data in an acceptable timeframe. This decrease in N may have diminished the reliability of the data. The likely source of the long run-times was the many structure arrays that were passed between, and altered by, many functions, which is an expensive operation.

This model of restaurant seating assumes that `revPASH` is the best indicator of performance. However, it might be important to consider other factors. Customer satisfaction is key in maintaining long-term success in the restaurant business. Sometimes the busiest nights, in which `revPASH` might be very high, result in a high number of dissatisfied customers, often due to long wait times and service mistakes that arise due to the high demand. Angry customers may then choose to not return, or worse, discourage others from dining there based on their bad experience. A more sophisticated model might consider this tradeoff between maximizing customer flow and customer satisfaction when addressing the seating problem.

Further Research

Continued exploration of the proposed decision-based model could reveal more conclusions about its performance. Repetition of the simulations performed in this project, but with a higher value of N might produce more reliable mean data for each value of T , with less variance. This could be done by running the same implementation with a larger N value on a machine with a more powerful processor, or re-implementing the approach in a different language and optimizing the data structures used to minimize the required operations and time complexity.

Another valuable extension of this model might be to have the decision-based approach consider seating in different sections, like the naïve one. A comparison of the two after this alteration might better reveal whether or not the decision-making process performs significantly better when their restaurant organization is the same.

Adding a queue walk in customers would be a simple solution to add flexibility to the model. Instead of declining walk in customers outright, they could be offered a place on a wait list, which is common practice in many restaurants. That way, as soon as a table opens up, they could be sat down. On busier nights, this approach could dramatically improve seat utilization, as the many rejected parties would be essentially a pool of demand that the restaurant could satisfy whenever supply becomes available. However, such a model introduces its own complexities, such as a customer declining to be placed on the waitlist, or waiting for an hour before deciding to dine elsewhere.

Experimentation with the decision quotient itself could provide enlightening results. Perhaps one could consider the probability of parties of a more appropriate size arriving in the next half hour, rather than hour. Alternatively, the numerator could be modified to only include the probability that a party of an exact fit will come in within the chosen time interval. Either of these modifications would change the factors that determine the decision, and could alter the outcomes.

It might be valuable to collect more specific restaurant data from Café Provence in order to refine the model. The choices of party size probabilities were primarily based on personal work experience, not data collection from the source. Kimes, Barrash, and Alexander discuss strategies for customer tracking electronically using software already in use by the restaurant, and human observation [1]. This might also allow one to consider the effectiveness of each seating approach for different days of the week or other meal times. After collecting more data, the decision model could be implemented at the restaurant by substituting random number generation with user input to produce customer lists. This would be an easy way to determine its effectiveness under actual restaurant conditions.

Integer programming, a more sophisticated optimization technique, is worth exploration as a method of solving the seating problem. While it was considered for this project, integer programming requires considerably more time and effort to implement – more than allotted for the project. With restaurant seating, the number of independent decision variables in an integer program increases rapidly with the size and complexity of the restaurant in question, as does the set of constraints imposed on the model. Another challenge of integer programming is that not all methods are guaranteed to work when solving them, especially when constraints and objective functions may be nonlinear. While MATLAB does have tools to solve both linear and nonlinear integer programs, they can be very time consuming, making them hard to test in a reasonable amount of time, especially if one is attempting to model customers in close to real time.

Conclusion

The decision-based approach to seating customers at a restaurant was found to outperform a more naïve method that sat customers by section, under certain conditions, when compared by their average revPASH values from a Monte Carlo simulation. The best conditions for the decision method were high customer demand and a large threshold value that was close to one. The model operated in real time, making a decision each time a customer made a request; such an approach could feasibly be implemented in a restaurant as a tool for hosts and managers.

Optimization of seating customers is of great interest to restaurant owners and managers desiring to improve the efficiency of their table use. In such a risk-filled business, even small improvements can take a restaurant from losing money to turning a profit. A solution to the seating problem might be of interest elsewhere in the scientific and engineering communities because it is an example of the larger class of scheduling problems, which require allocating resources to complete tasks. A method that performs well for the seating could be adapted and applied to other scheduling problems. These could be in other service industries, such as the booking of hotels, or in technological applications like managing the power needs of the grid or directing client requests to a collection of web servers.

References

- [1] S.E. Kimes, D.L. Barrash, J.E. Alexander, *Developing a Restaurant Revenue-Management Strategy*, Cornell Hotel and Restaurant Administration Quarterly, 40 (1999), pp. 18-29.
- [2] A. Vidotto, *Managing Restaurant Tables Using Constraint Programming*, PhD Thesis, National University of Ireland, 2007.
- [3] J. Feldman, *Optimizing Restaurant Reservation Scheduling*, Thesis, Harvey Mudd College, 2010.

Average Covers: 60 Frac Res: 0.30

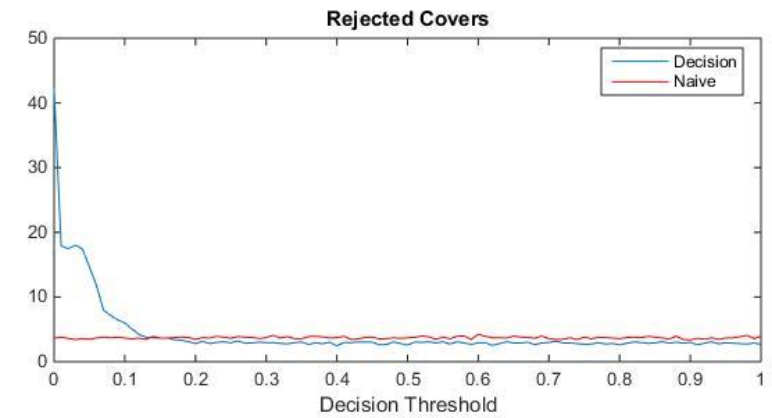
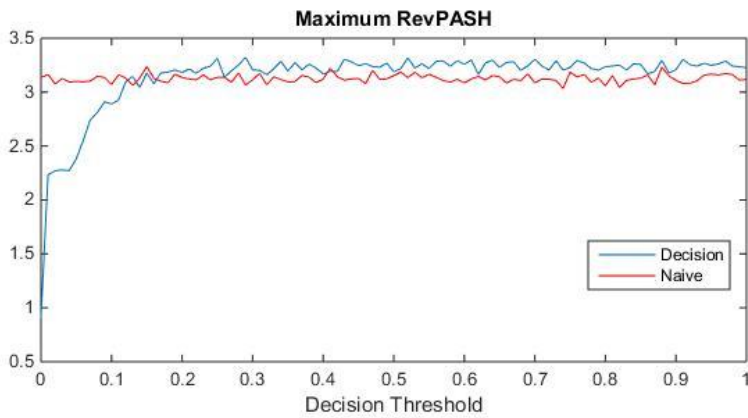
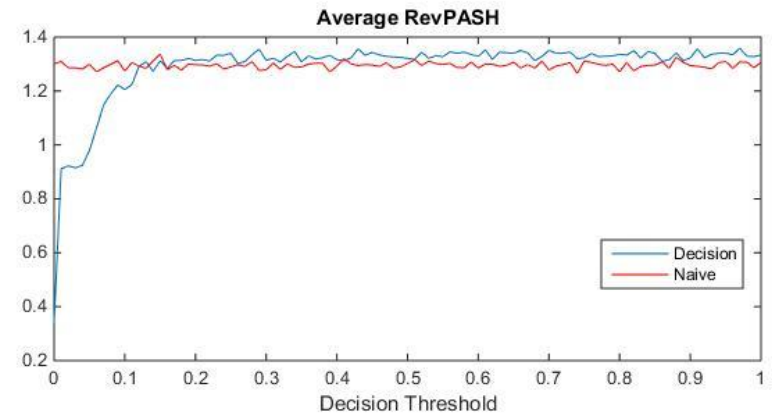
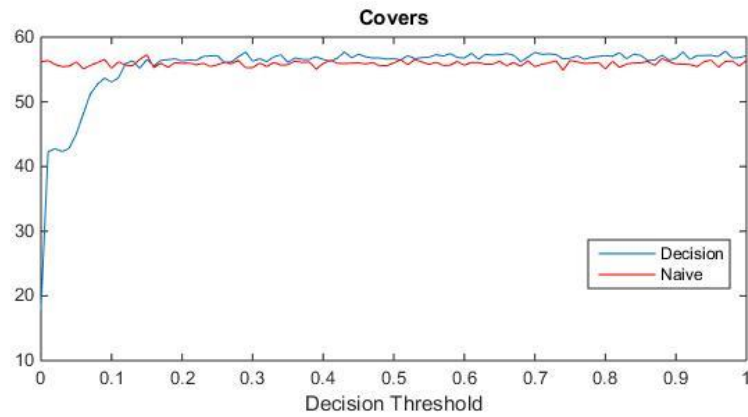


Figure 1. Performance of decision-based and naïve seating algorithms with average nightly covers of 60 and the fraction of reservations of 0.3

Average Covers: 80 Frac Res: 0.30

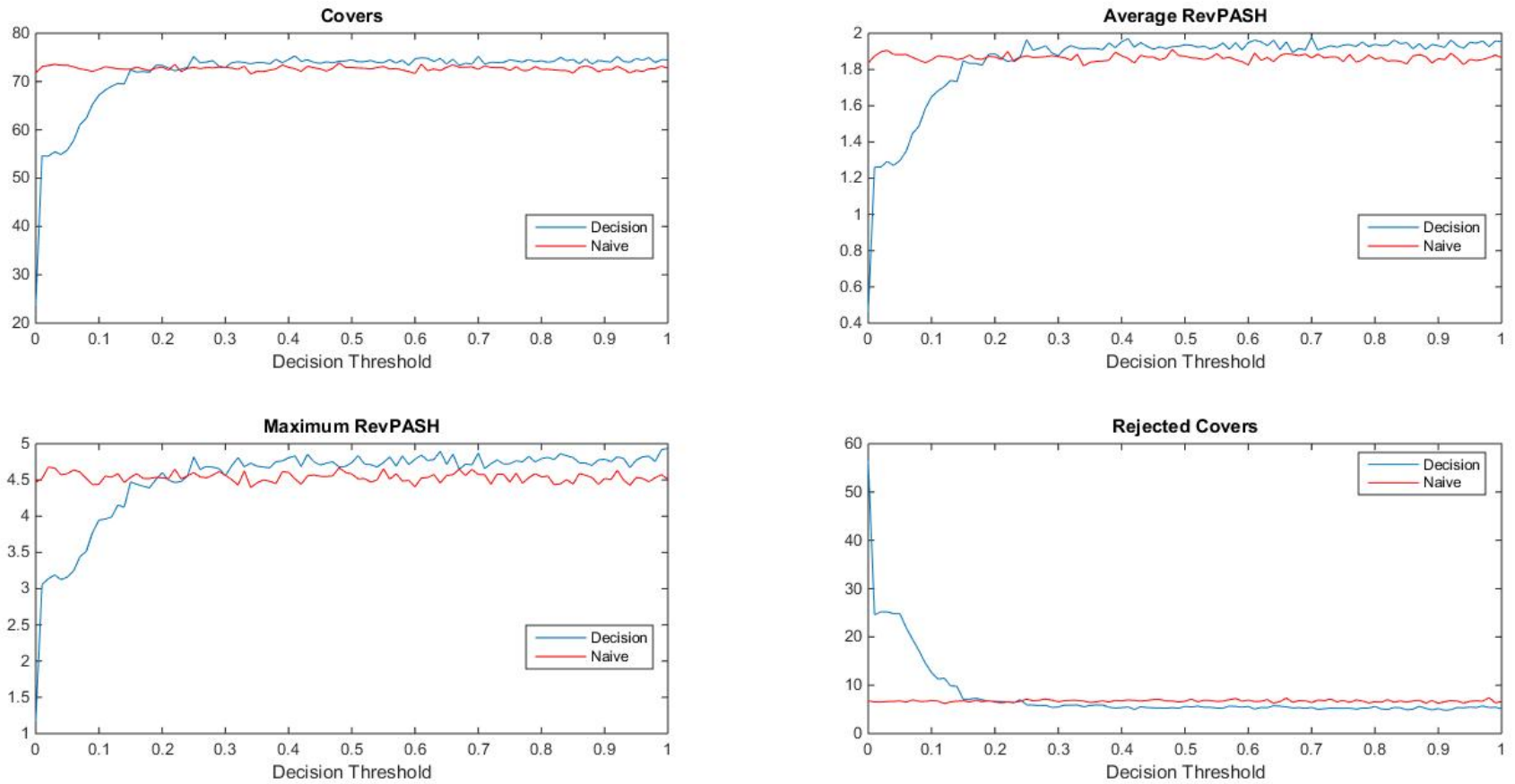


Figure 2. Performance of decision-based and naïve seating algorithms with average nightly covers of 80 and the fraction of reservations of 0.3

Average Covers: 100 Frac Res: 0.30

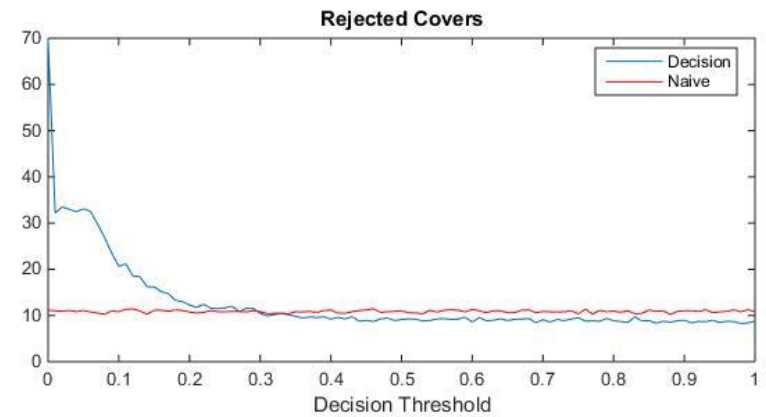
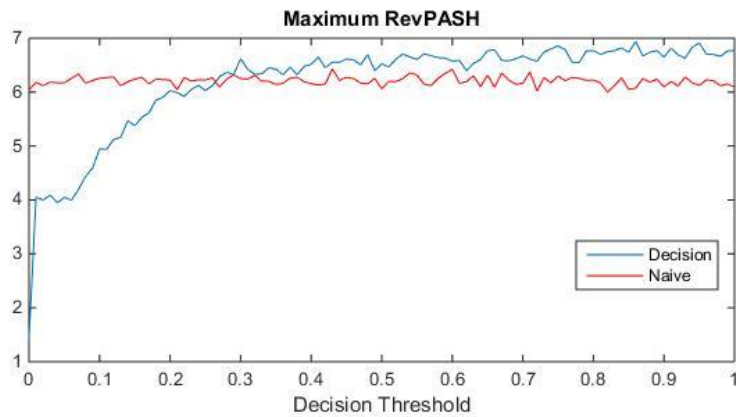
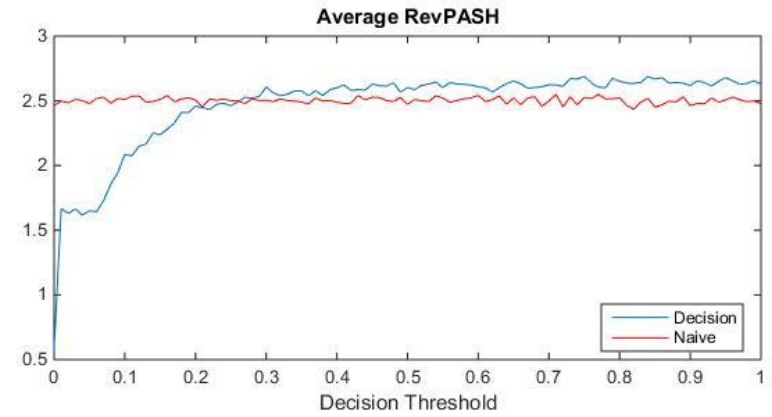
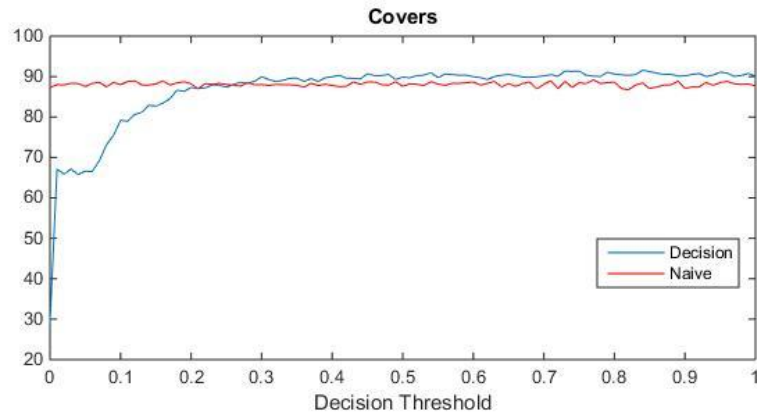


Figure 3. Performance of decision-based and naïve seating algorithms with average nightly covers of 100 and the fraction of reservations of 0.3

Average Covers: 111 Frac Res: 0.30

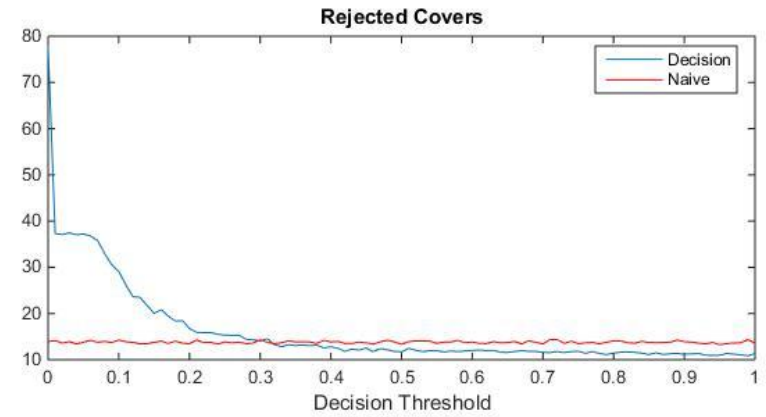
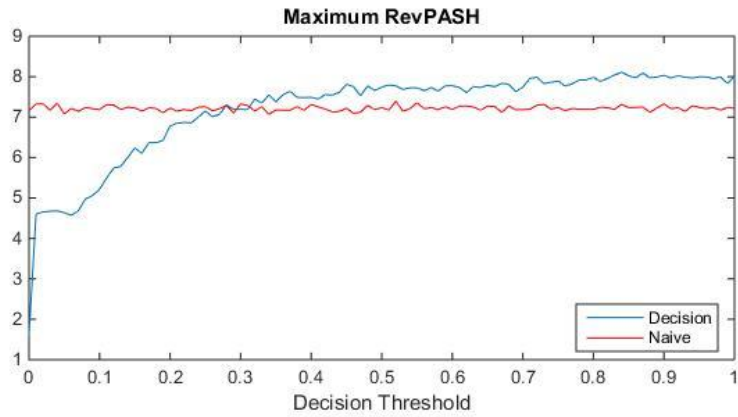
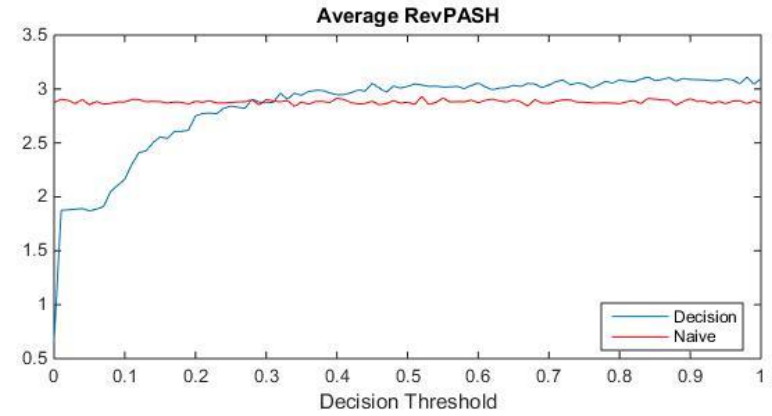
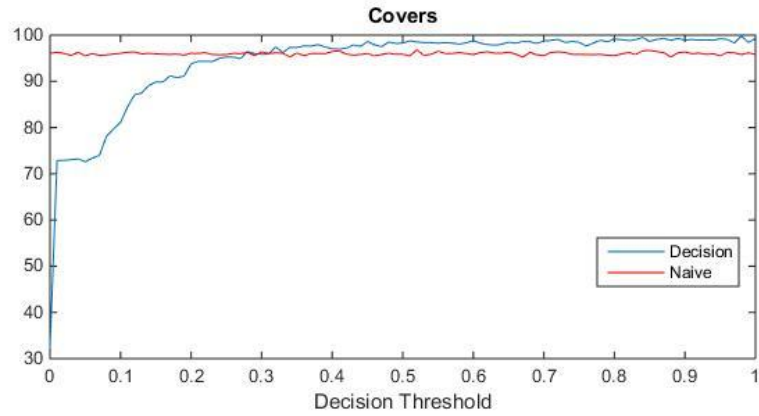


Figure 4. Performance of decision-based and naïve seating algorithms with average nightly covers of 111 and the fraction of reservations of 0.3

Average Covers: 120 Frac Res: 0.30

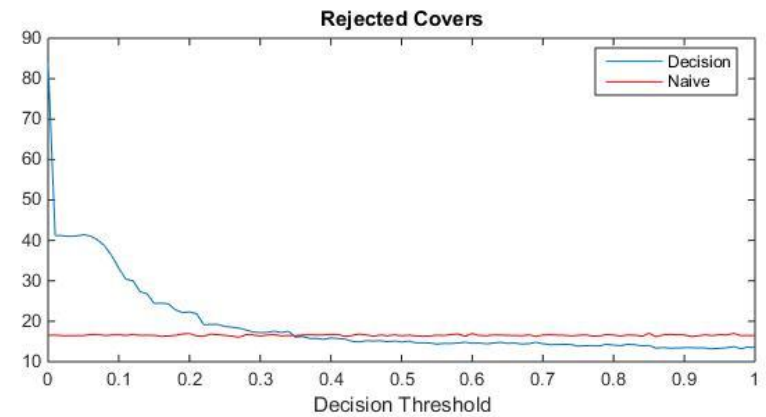
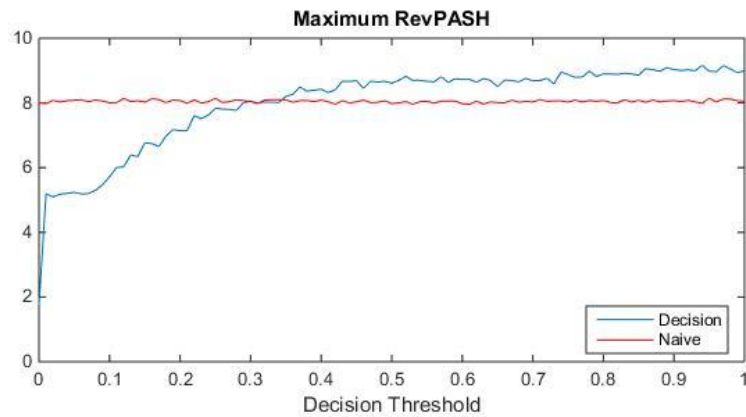
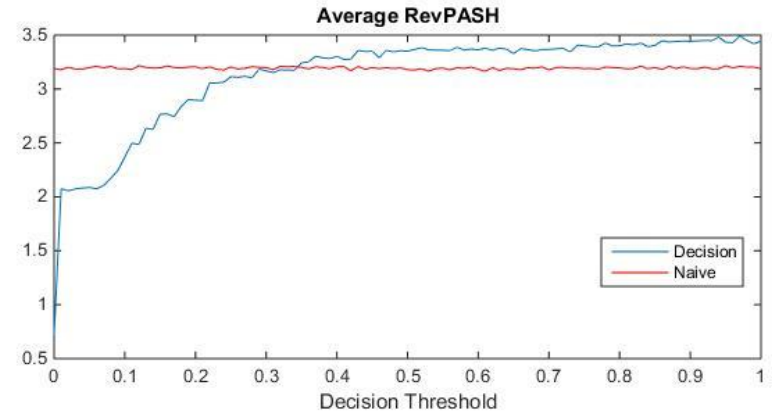
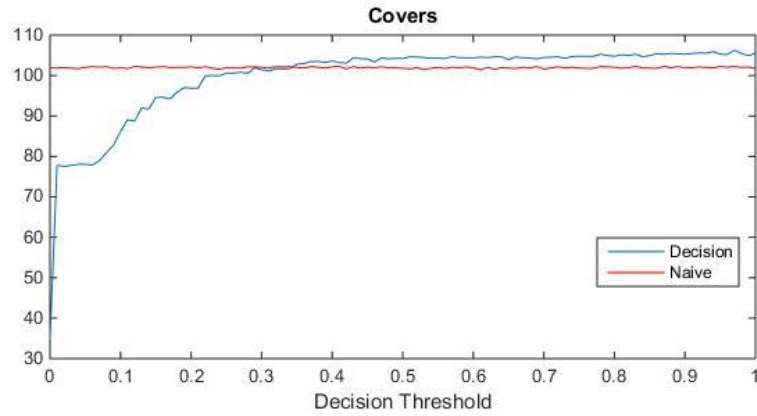


Figure 5. Performance of decision-based and naïve seating algorithms with average nightly covers of 120 and the fraction of reservations of 0.3

Average Covers: 140 Frac Res: 0.30

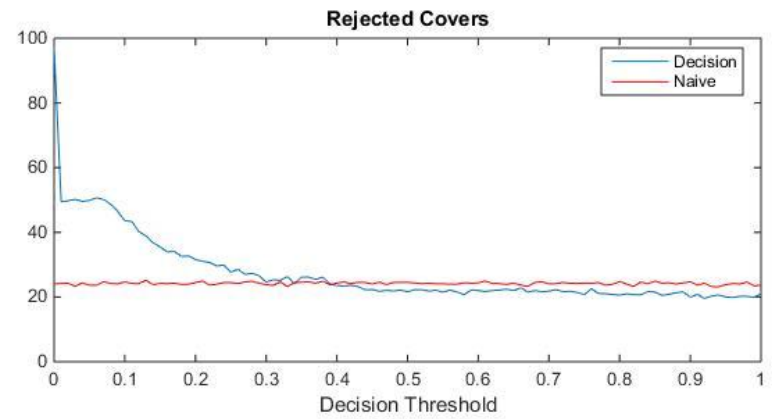
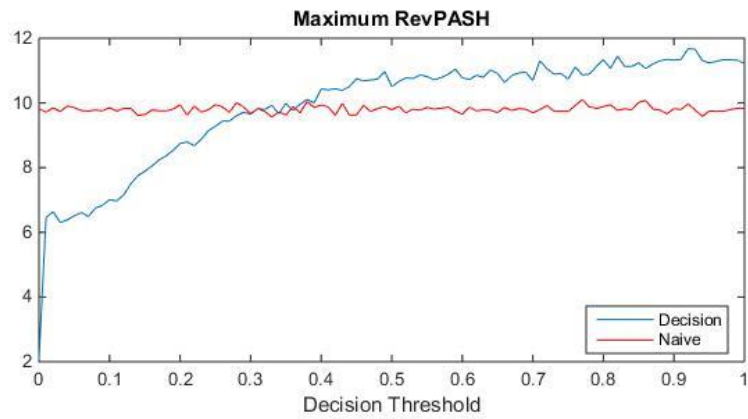
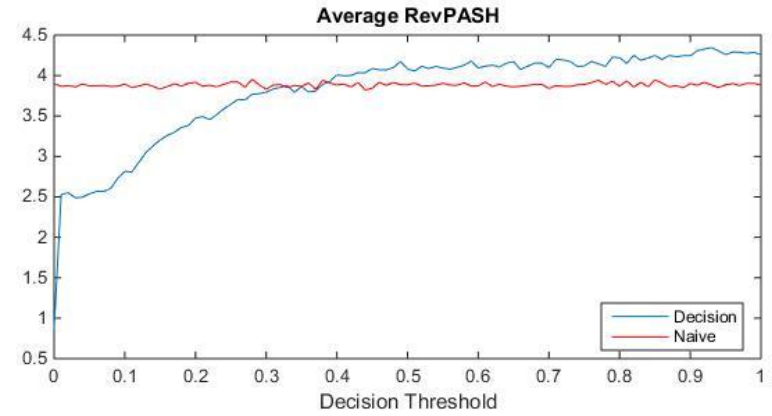
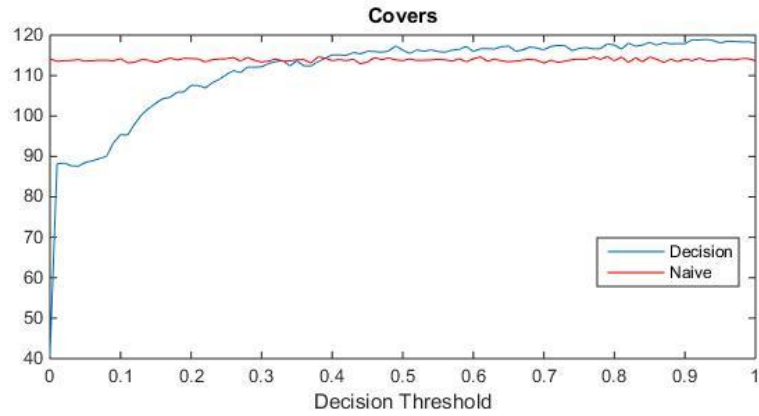


Figure 6. Performance of decision-based and naïve seating algorithms with average nightly covers of 140 and the fraction of reservations of 0.3

Average Covers: 160 Frac Res: 0.30

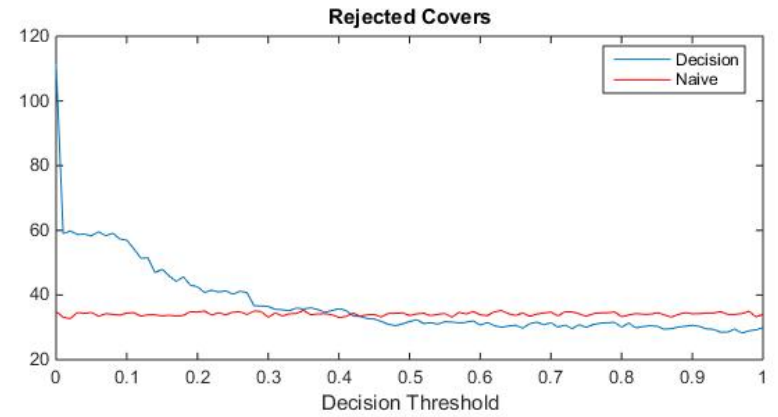
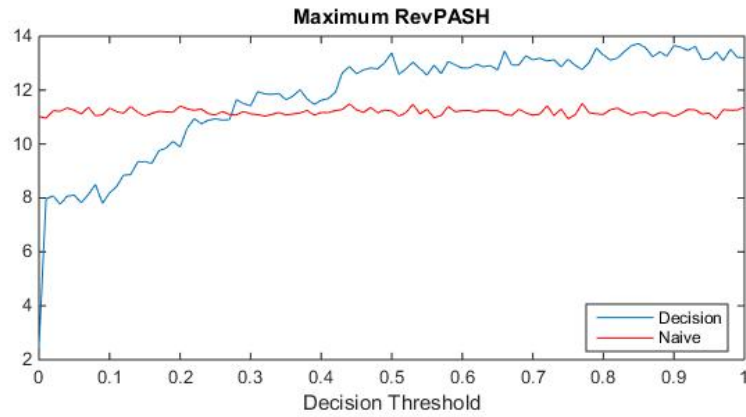
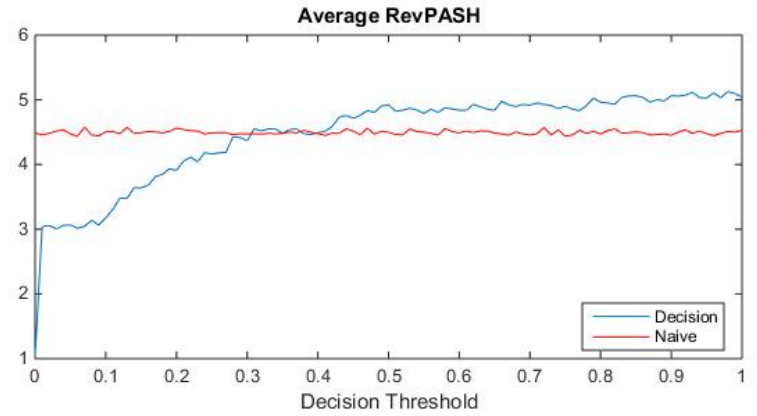
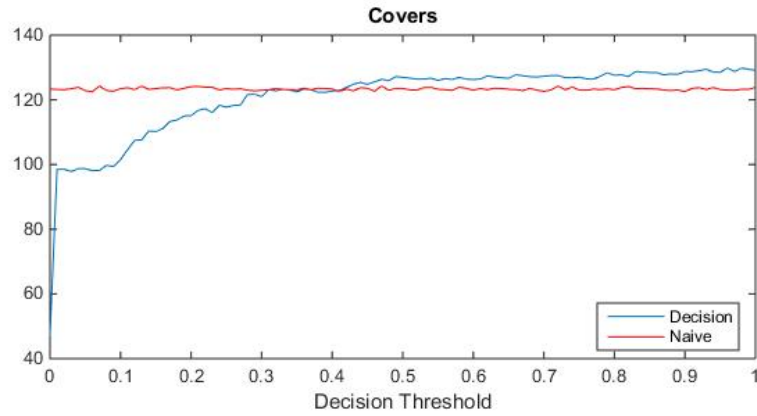


Figure 7. Performance of decision-based and naïve seating algorithms with average nightly covers of 160 and the fraction of reservations of 0.3