시뮬레이션

안 암 고 른 햇 살 분 석

2015170832 이남경

2015170838 우성훈

2017170823 이예진

목차

I. Introduction

1. Target System Description

Figure 1. 대표메뉴

Figure 2. 식사시간의 흔한 대기행렬

고른햇살은 안암을 대표하는 분식집으로 많은 고려대학교 재학생 및 인근 거주민들이 이용하고 있는 음식점이다. 밥보다 참치가 많은 듯한 팔뚝만한 참치 김밥과 치즈사리 라볶이가 대표메뉴이며 모든 메뉴가 1500 ~ 4000원의 비교적 저렴한 가격이다. 맛과 양 두 마리 토끼를 모두 잡은 탓에 점심, 저녁시간 고른햇살은 학생들로 문전성시를 이룬다. 그로 인하여 고른햇살은 식사시간대에 가게 앞에 매우 긴 대기행렬이 발생한다. 우리는 분식업계 특성상 조리시간이 짧고, 이용고객의 식사시간이 짧음에도 불구하고 현재와 같이 긴 대기행렬이 발생함을 확인하였고 그에 따라 고른햇살의 현재 system에 문제가 있다고 판단, 이를 개선하기 위한 새로운 운영 system을 제안하고자 한다. 개선된 고른햇살 운영 system을 통하여 많은 학생들이 저렴하게 한 끼를 해결하고 고른햇살 사장님의 수익의 극대화하여 win-win을 이루고자 한다.

2. Necessity of system improvement

우리는 대기행렬을 줄이기 위한 고른햇살의 system 개선을 학생들이 원하고 있음을 인터뷰를 통해 확인하였다. 설문조사는 총 6 명에 대하여 진행하였다.

i. 대상 : 고른햇살을 점심시간에 이용하지 않는 학생

Question 1	점심식사 시간에 고른햇살을 이용하지 않는 이유가 있는가?
A(24세,남)	점심시간에 줄이 너무 길다. 고른햇살 먹으려면 뛰어가야한다. 뛰어가서 먹을정도로 먹고싶진 않다.
B(21세,여)	매장에서 식사하기엔 줄이 길어서 다른 것을 먹거나 포장하는 편이다.
C(26세,남)	밥 혼자 먹을 때가 많아서 줄이 길면 눈치보여서 안간다.

표 1. 점심시간 고른햇살 선호도 조사

Question 2	고른햇살의 줄이 줄어든다면 이용할 의향이 있는가?
A(24세,남)	매일은 아니지만 일주일에 2번정도는 이용할 의향이 있다.
B(21세,여)	지금은 줄이 길어서 아예 고른햇살에서 밥먹을 생각을 안하지만 줄이 준다면 자주 고려하게 될 것 같다.
C(26세,남)	혼잡도가 줄어 혼밥하기 편한 분위기가 된다면 애용할 것 같다.

표 2. 고른햇살 시스템 개선 시 선호도 조사

ii. 대상 : 고른햇살 앞에서 대기행렬로 인하여 balking한 학생

Question 1	고른햇살을 이용하지 않고 왜 그냥 가는가?
D(21세,남)	줄이 너무 길다. 친구 4명이서 왔는데 먹기엔 너무 좁다
E(24세,여)	기다리기엔 너무 춥다.
F(22세,여)	시간이 부족해서 기다릴수 없다.

표 3. 대기행렬로 인한 고른햇살 balking 발생

우리는 설문을 통하여 고른햇살의 balking은 가게 앞에서만 이루어 지는 것이 아니라 많은 학생들이 고른햇살의 식사시간대는 줄이 길다는 것을 이미 인식하고 방문하지 않는 경우가 많았다. 그러나 많은 학생들이 고른햇살의 대기행렬이 개선된다면 방문하고 싶다는 의사를 밝히면서 고른햇살의 system 개선을 원하고 있었다. 우리는 비단 사장님의 매출증대 뿐만 아니라 배고픈 학생들의 주머니 사정을 위해서라도 고른햇살의 시스템 개선은 필요하다고 판단하여 simulation project의 주제로 선정하였다.

3. Purpose of simulation

본 시뮬레이션의 목적을 요약하면 다음과 같다. 본 시뮬레이션의 목적을 '고른햇살 테이블 배치 재구성을 통한 대기행렬 감소'이다. 현재 고른햇살의 테이블 배치는 4인 테이블 6개로 구성되어 있다. 우리는 4인 테이블을 제거하고 2인 테이블을 추가하는 방식으로 system을 개선하고자 한다. 우리는 빠르고 간편하게 식사를 해결하고자 하는 분식업계 및 점심시간 특성 상 1,2인 손님의 비율이 높을 것이라 예상하고 현재 1,2인 손님이 4인 테이블을 이용할 경우 발생하는 공석에 대한 비효율성을 줄이고자 한다.

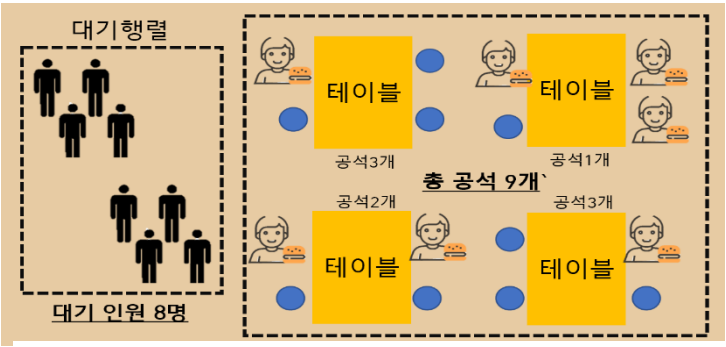


Figure 3. 4인 테이블의 비효율성

그러나 2인테이블 2개는 4인테이블 1개보다 많은 공간을 차지하므로 우리는 2인 테이블로 4을 테이블 교체 시 발생하는 공간제약을 고려하고자 한다. 이와 같은 개선방식의 장점으로는 3가지가 있다.

1	System 개선을 위하여 고른햇살의 종업원수의 조정을 위한 추가비용이 발생하지 않는다.
2	매출이 상승할 것이다.
3	2인 테이블 도입으로 인하여 좁은 식당 내부공간이 쾌적해질 것이다.

표 4. 시뮬레이션 개선방식의 장점

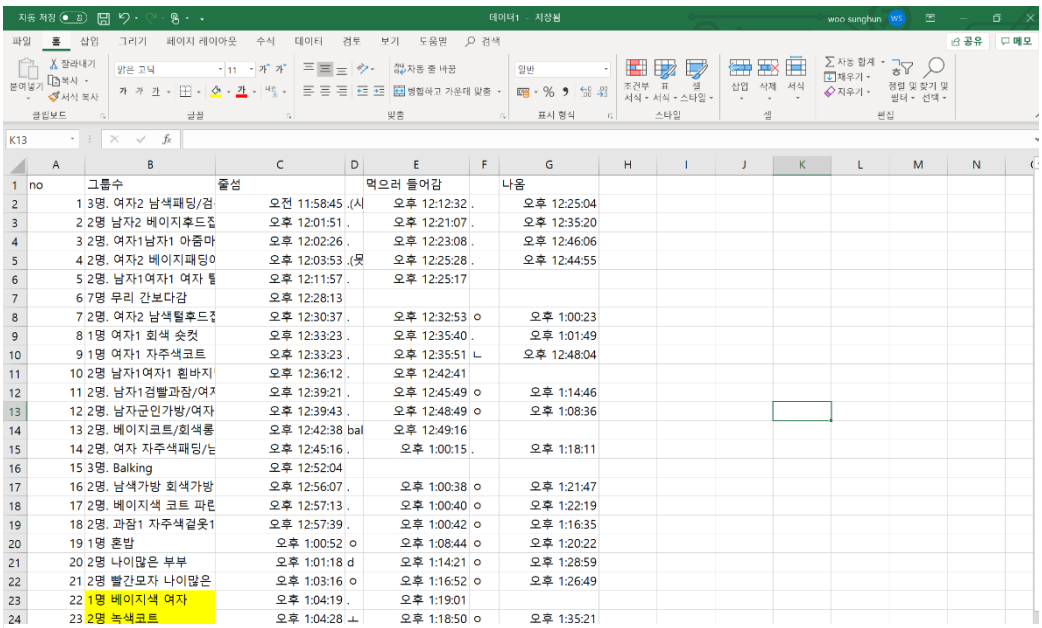
II. Data Analysis

1. 데이터 수집

우리는 11월 11~14일, 총 4일, 점심시간 11:30~13:30분을 대상으로 고른햇살 고객 데이터를 수집하였다. 금요일이 공강인 학생들이 많다고 예상되어 금요일에 고른햇살을 방문하는 손님이 다른 요일에 비하여 적을 것이라 판단하였다. 그에 따라, 월,화,수,목 4일에 대하여 데이터를 수집하였다.

수집기간	2019년 11월 11일~14일(4일) 12:00 ~ 14:00
수집대상	고른햇살에서 점심식사를 해결하고자 하는 고객
고객유형	1,2,3,4인으로 구성된 그룹과 그 외 기타 (포장제외)
Data 유형	각 그룹별 interarrival time, service time, balking rate, queue의 최대 길이

표 5. 데이터 수집 계획



no	그룹수	줄섬		먹으러 들어감		나옴
1	3명. 여자2 남색패딩/검	오전 11:58:45	.(시	오후 12:12:32	.	오후 12:25:04
2	2명 남자2 베이지후드집	오후 12:01:51	.	오후 12:21:07	.	오후 12:35:20
3	2명. 여자1남자1 아줌마	오후 12:02:26	.	오후 12:23:08	.	오후 12:46:06
4	2명. 여자2 베이지패딩이	오후 12:03:53	.(뭇	오후 12:25:28	.	오후 12:44:55
5	2명. 남자1여자1 여자 틀	오후 12:11:57	.	오후 12:25:17		
6	7명 무리 간보다감	오후 12:28:13				
7	2명. 여자2 남색털후드집	오후 12:30:37	.	오후 12:32:53	o	오후 1:00:23
8	1명 여자1 회색 숏컷	오후 12:33:23	.	오후 12:35:40	.	오후 1:01:49
9	1명 여자1 자주색코트	오후 12:33:23	.	오후 12:35:51	ㄴ	오후 12:48:04
10	2명 남자1여자1 흰바지	오후 12:36:12	.	오후 12:42:41		
11	2명. 남자1검빨과잠/여자	오후 12:39:21	.	오후 12:45:49	o	오후 1:14:46
12	2명. 남자군인가방/여자	오후 12:39:43	.	오후 12:48:49	o	오후 1:08:36
13	2명. 베이지코트/회색롱	오후 12:42:38	bal	오후 12:49:16		
14	2명. 여자 자주색패딩/남	오후 12:45:16	.	오후 1:00:15	.	오후 1:18:11
15	3명. Balking	오후 12:52:04				
16	2명. 남색가방 회색가방	오후 12:56:07	.	오후 1:00:38	o	오후 1:21:47
17	2명. 베이지색 코트 파른	오후 12:57:13	.	오후 1:00:40	o	오후 1:22:19
18	2명. 과잠1 자주색겉옷1	오후 12:57:39	.	오후 1:00:42	o	오후 1:16:35
19	1명 혼밥	오후 1:00:52	o	오후 1:08:44	o	오후 1:20:22
20	2명 나이많은 부부	오후 1:01:18	d	오후 1:14:21	o	오후 1:28:59
21	2명 빨간모자 나이많은	오후 1:03:16	o	오후 1:16:52	o	오후 1:26:49
22	1명 베이지색 여자	오후 1:04:19	.	오후 1:19:01		
23	2명 녹색코트	오후 1:04:28	ㅗ	오후 1:18:50	o	오후 1:35:21

Figure 4. 수집된 데이터

데이터 수집 방식은 고른햇살 건너편 아마스빈에서 노트북을 이용하여 실시간으로 데이터를 입력하였다. 이때 엑셀 함수를 이용하여 고객의 수를 입력하면 자동으로 시간이 들어가도록 설계를 하여 데이터의 정확도를 올리고자 노력하였다.

2. Input Data

수집한 데이터는 다음과 같다.

	Interarrival					Service Time
	1명	2명	3명	4명	4명이상	
표본 수	36	124	35	4	1	156
평균(sec)	496.983	277.266	775.5706	1289.5	N/A	1143.8

표 6. 수집된 데이터 요약

데이터 수집 결과 4명으로 이루어진 그룹의 손님수의 표본이 너무 적어 손님의 interarrival time에 대한 분포 추정은 1,2,3인 그룹에 대하여 진행하고, 4인 그룹의 경우 3인 그룹의 일부로 편입시켰다. 각 그룹별 Service Time은 같은 분포를 가진다고 가정하고 추정하였다.

3. 통계적 분포 추정

분포 추정을 위한 절차는 아래와 같이 6가지 단계를 거친다.

	내용	방법
Step 1	Homogeneity Check	Kruskal-Wallis test
Step 2	Independence Check	Scatter-plot, Correlation plot
Step 3	Distribution Families Estimation	Histogram, Quantile summary, Box plot
Step 4	Distribution Parameter Estimation	EASYFIT-program(MLE Method)
Step 5	Distribution & Parameter Validation	CI Sensitivity Analysis
Step 6	Fitted Distribution Validation	DFD plot, P-P plot, Q-Q plot, EASYFIT-program(Goodness-of-Fit test)

표 7. 분포 추정 절차

STEP 1) Homogeneity Check

모든 데이터는 4일간 고른햇살에 손님이 몰리는 점심시간(12:00 ~ 14:00)동안 거의 동일한 상황에서 총 4회 수집되었다. 데이터의 통계 분포를 추정하기 위하여 많은 표본이 요구된다. 따라서 수일간에 걸쳐 수집된 데이터가 동질성을 갖는지 검증하고, 검증 결과가 동질하다면 데이터를 합쳐서 보다 많은 표본을 이용하여 분포를 추정할 수 있다.

데이터의 동질성(Homogeneity)를 검증하는 방법으로, Kruskal-Wallis Chi-square 검정을 실시하였다.

H0	All the population distribution functions are identical
H1	At least one of the populations tends to yield larger observations than at least one of the other populations

검정을 위한 통계프로그램으로는 R을 사용하였다.

	Interarrival			Service time
	1인	2인	3인	
p-value	0.1359	0.4547	0.154	0.7193
Result	Fail to reject H0	Fail to reject H0	Fail to reject H0	Fail to reject H0

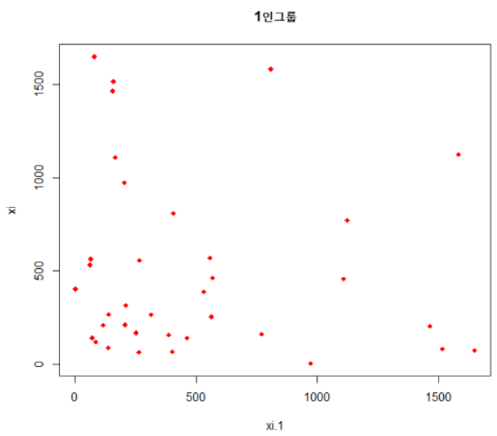
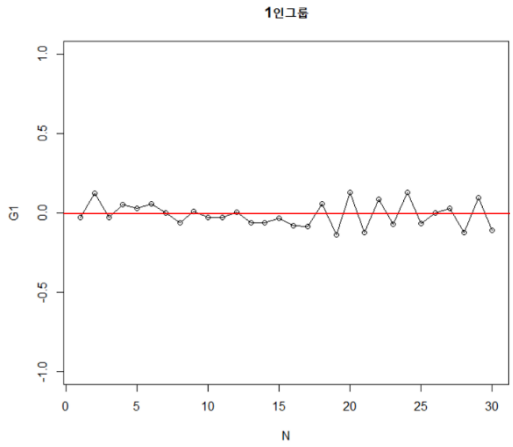
표 8. Homogeneity Check 결과

데이터 동질성 검정 결과 수집된 데이터의 p-value>0.05 이므로 모두 0.05의 유의수준에서 통계적으로 귀무가설을 기각할 수 없다. 따라서 분포추정을 위하여 수집된 다른 요일의 데이터를 합쳐서 사용할 수 있다.

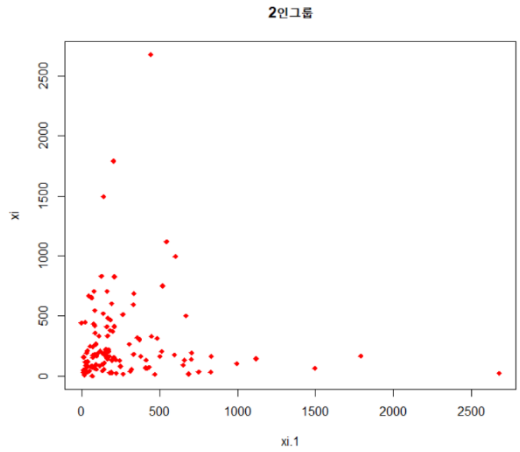
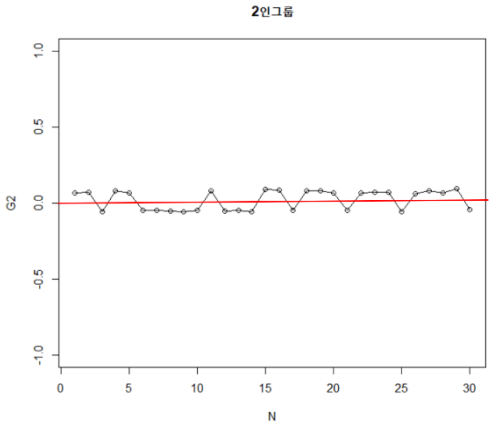
Step 2) Independence Check

시뮬레이션의 input data는 동일한 분포로 가정할 수 있는 서로 독립인 데이터일 때 이후의 통계적 분석이 가능하다. 독립성을 확인하기 위하여 Heuristic procedure의 일환으로 1) Scatter-plot, 2) Correlation-plot을 이용한다.

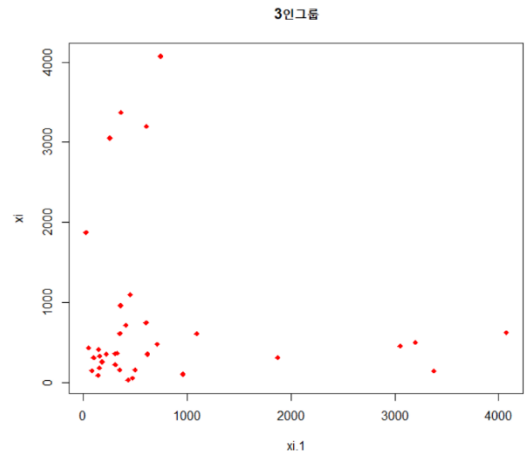
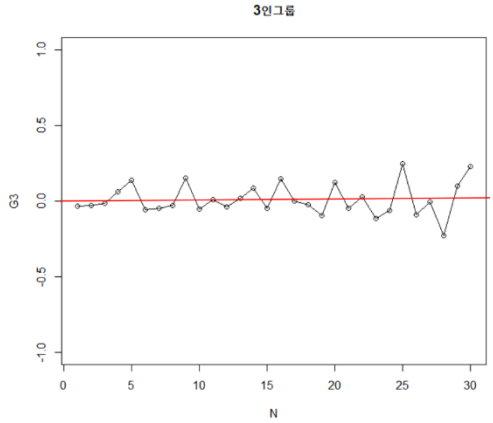
①. 1인 그룹 interarrival

Scatter-plot	Correlation-plot
	

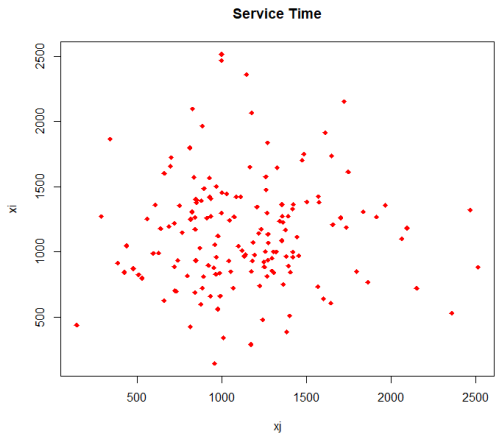
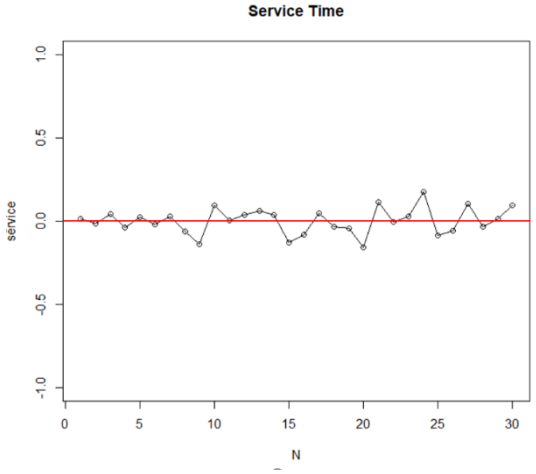
②. 2인 그룹 interarrival

Scatter-plot	Correlation-plot
	

③. 3인그룹 interarrival

Scatter-plot	Correlation-plot
	

④. service time

Scatter-plot	Correlation-plot
	

결론

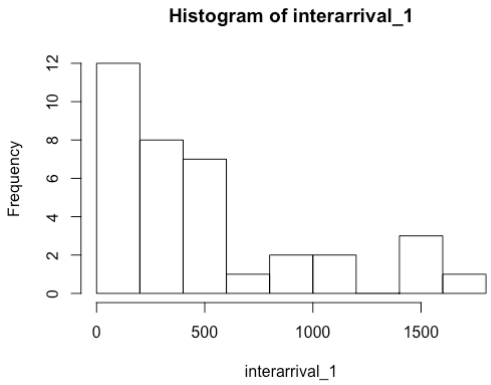
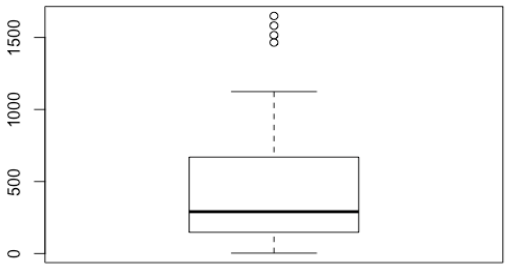
1) Scatter plot을 확인한 결과 data의 분포가 특정한 패턴을 나타내지 않았다

2) Correlation plot을 확인한 결과 data의 상관관계가 나타나지 않았다.

따라서 Heuristic procedure를 통하여 4개의 분포에 대한 data의 independency를 확인하였다.

Step 3) Distribution Families Estimation

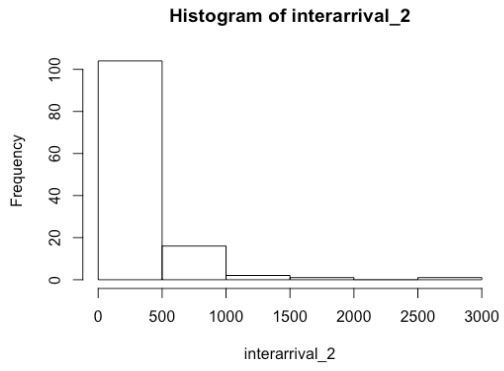
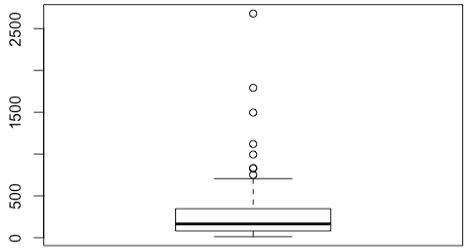
①. 1인 그룹 interarrival

Histogram	Box Plot
	

Quantile Summaries					
Min	1st Qu.	Median	Mean	3rd Qu.	Max
3.0	151.8	290.5	497.0	618.7	1649.0

1인 그룹 interarrival time의 분포 개형 추정 결과 좌편향(skewed left) 되어 있는 분포임을 확인할 수 있다.

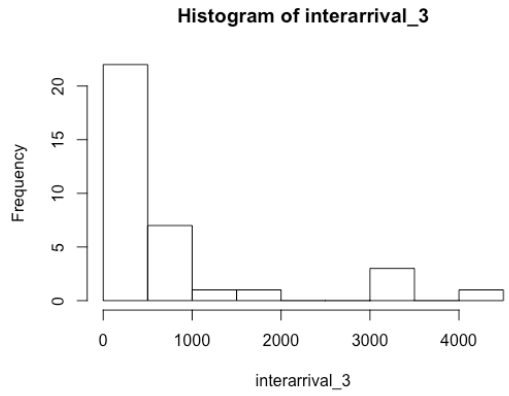
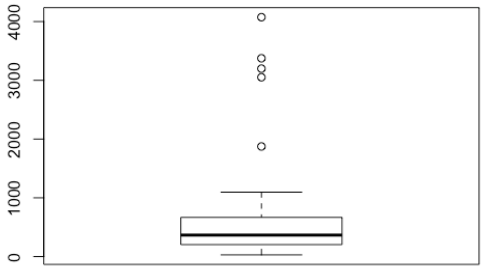
②. 2인 그룹 interarrival

Histogram	Box Plot
	

Quantile Summaries					
Min	1st Qu.	Median	Mean	3rd Qu.	Max
12.0	81.5	164.8	280.3	340.7	2679.3

2인 그룹 interarrival time의 분포 개형 추정 결과 좌편향(skewed left) 되어 있는 분포임을 확인할 수 있다.

③. 3인 그룹 interarrival

Histogram	Box Plot
	

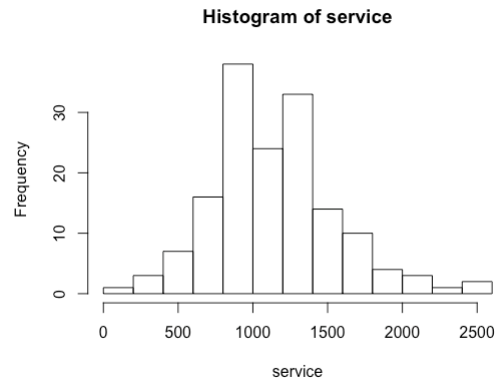
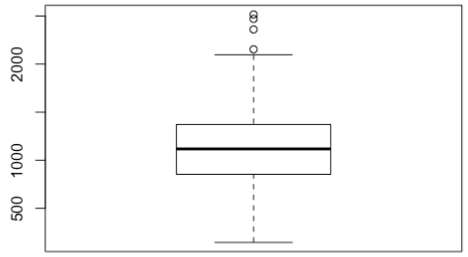
Quantile Summaries					
Min	1st Qu.	Median	Mean	3rd Qu.	Max
29.0	203.8	366.0	775.6	667.0	4074.2

3인 그룹 interarrival time의 분포 개형 추정 결과 좌편향(skewed left) 되어 있는 분포임을 확인할 수 있다.

결론

그룹별 interarrival time 분포의 평균을 확인해 보면 3인 > 1인 > 2인 임을 확인할 수 있다. 이는 고른햇살을 이용하는 그룹별 interarrival time 간격이 1,2인 그룹이 짧고 3인 그룹이 긴 것을 의미한다. 따라서 고른햇살에는 1,2명의 그룹으로 이루어진 손님들이 더 많이 올 것이라고 예상할 수 있다.

④. Service time

Histogram	Box Plot
	

Quantile Summaries

Min	1st Qu.	Median	Mean	3rd Qu.	Max
145.0	853.8	1117.6	1143.8	1349.0	2156.0

Service Time의 분포 개형을 확인해 보면 대칭적 구조에서 약간 좌편향(skewed left)되어 있음을 확인할 수 있다. interarrival time의 분포 보다는 덜 좌편향 되어 있음을 확인할 수 있다.

Step 4) Distribution Parameter Estimation

통계 소프트웨어 EASYFIT을 이용하여 interarrival 3개의 분포, service time 1개의 분포에 대하여 수집한 data와 가장 유사한 분포를 추정하였다. EASYFIT의 분포 추정방법은 Maximum-Likelihood-Estimation(MLE) method이며 우리는 이를 통하여 각 분포의 형태와 parameter를 얻었다. 그 결과는 아래와 같다.

Input Data		분포	Estimator		
			Parameter1	Parameter2	Parameter3
Interarrival	1인 그룹	Weilbull(2p)	0.89876	501.58	
	2인 그룹	Log-Logistic(2p)	1.6152	163.13	
	3인 그룹	Frechet(3p)	1.4704	390.16	-112.21
Service Time		Gumbel Max	326.03	955.59	

Step 5) Distribution Parameter Validation

얻어진 Parameters들이 유효한것인지 검증해야한다. 'r fitdistrplus' 패키지를 통해 Parameters들의95% 신뢰구간을 얻었으며, 신뢰구간의 최댓값과 최솟값을 정리하면 아래와 같다.

		분포	Estimator								
			Parameter 1			Parameter 2			Parameter 3		
Interarrival	1인 그룹	Weilbull(2p)	0.768	0.899	1.171	414.59	501.58	592.42			
	2인 그룹	Log-Logistic(2p)	1.369	1.615	1.852	132.10	163.13	194.12			
	3인 그룹	Frechet(3p)	1.441	1.470	1.500	377.16	390.16	403.16	-121.37	-112.21	-103.05
Service Time		Gumbel Max	308.95	326.03	431.53	885.09	955.59	1007.96			

분포에 사용된 Parameter의 민감도 분석을 통해 추정한 Parameter들을 검증한다. 각각의 parameter들에 대한 신뢰구간에서의 최댓값과 최솟값을 Input parameter로 사용하여 각각 시뮬레이션을 시행하는 것이다. 이때 두 시뮬레이션의 출력값에 유의미한 차이가 없다면, 추정된 Parameter를 사용하고, 그렇지 않다면 데이터를 보충하여 Parameter를 다시 추정하여야 한다.

4개 분포의 parameter의 min, max 조합으로 simulation을 n=30씩 진행하였다. 1인, 2인, service의 분포는 parameter가 2개이므로 4개의 조합에 대해 시뮬레이션을 진행하였고, 3인 그룹의 분포는 parameter가 3개이므로 8개의 조합에 대해 진행하였다.

ANOVA를 통한 statistic 비교는 Policy 선정에 영향을 미치는 performance measure 로 average waiting time in queue에 대하여 행하였다.

H0	분포 A의 parameter1, 2의 조합이 [min, min], [min, max], [max, min], [max, max] 인 4가지 경우의 statistic 값이 동일하다.
H1	하나라도 다른 것이 존재한다.

***분포의 parameter가 2개인 경우**

H0	분포 A의 parameter1, 2, 3의 조합이 [min, min, min], [min, min, max]. [min, max, min], [min, max, max], [max, min, min], [max, min, max], [max, max, min] [max, max, max] 인 8가지 경우의 statistic 값이 동일하다.
H1	하나라도 다른 것이 존재한다.

***분포의 parameter가 3개인 경우**

이러한 방식으로 4개의 분포에 대해 진행하였으며, 그 결과로 alpha=0.05 수준에서 귀무가설은 기각되지 않았다. 따라서 구해진 parameter를 사용하였다.

Step 6) Fitted Distribution Validation

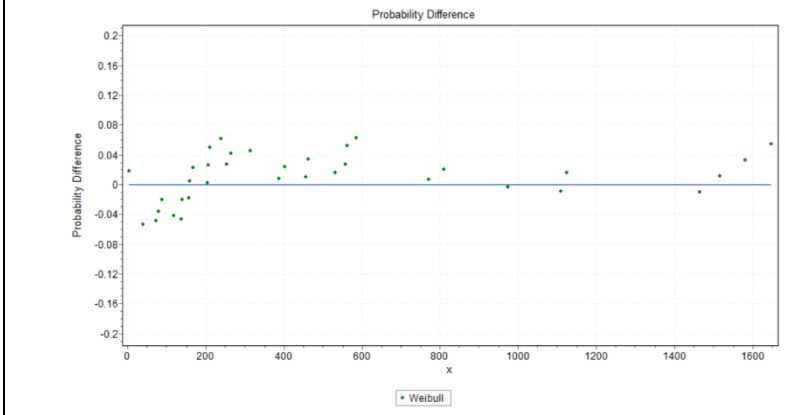
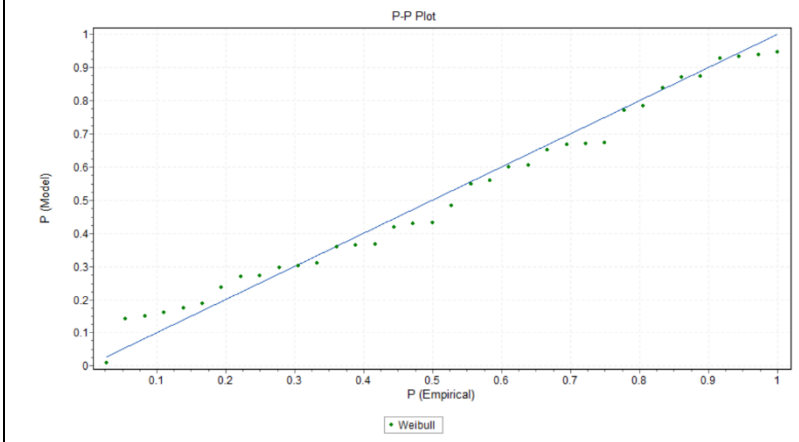
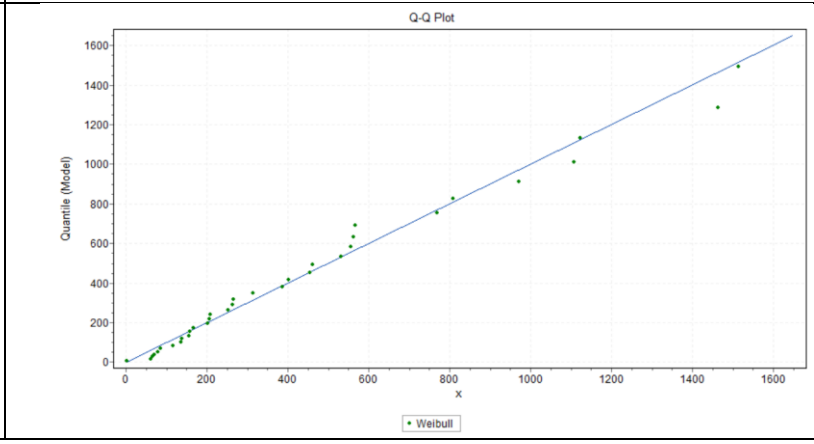
추정한 분포와 실제 데이터와 적합한지 확인하기 위하여 두가지 방법론을 고려하였다.

1) Heuristic Procedure – DFD plot, Q-Q plot, P-P plot
2) Statistical hypothesis tests – Chi-Square test, Kolmogorov-Smirnov test

총 5가지 방법에 대하여 검정을 진행하였으며 그 결과는 아래와 같다.

① Heuristic Procedure

A. 1인 그룹 Interarrival

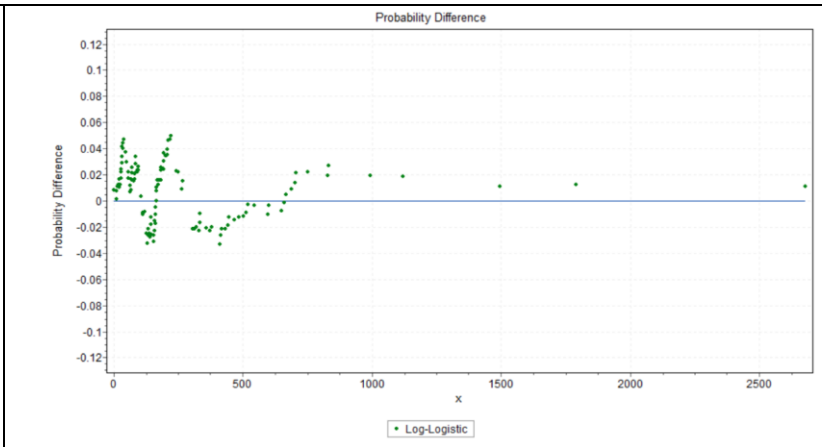
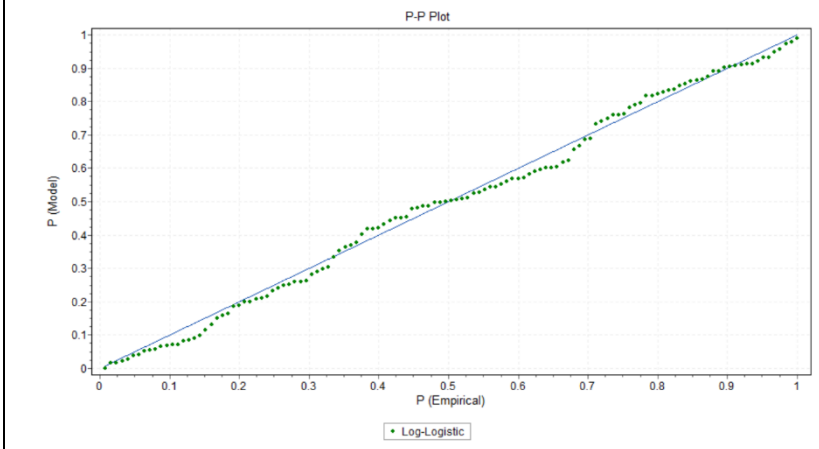
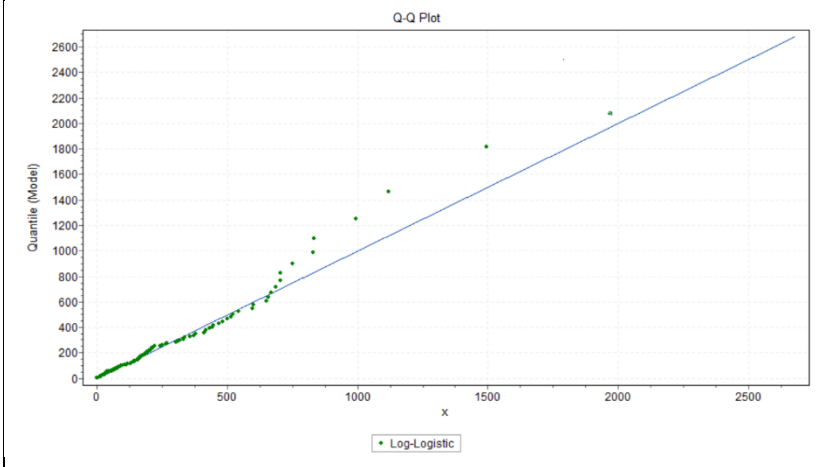
Heuristic Procedure	
DFD plot	
P-P plot	
Q-Q plot	

1) DFD plot : fitted distribution의 CDF와 input data를 기반으로 작성한 empirical distribution의 CDF의 차이가 0.05이상 크기 차이나는 지점이 거의 없다.

2) P-P plot : 분포 fitting의 중간 부분의 차이를 확인하기 위한 P-P plot을 확인한 결과 매우 큰 차이를 보이지 않았다.

3) Q-Q plot : 분포 fitting의 양 끝 부분의 차이를 확인하기 위한 Q-Q plot을 확인한 결과 매우 큰 차이를 보이지 않았다.

B. 2인 그룹 interarrival

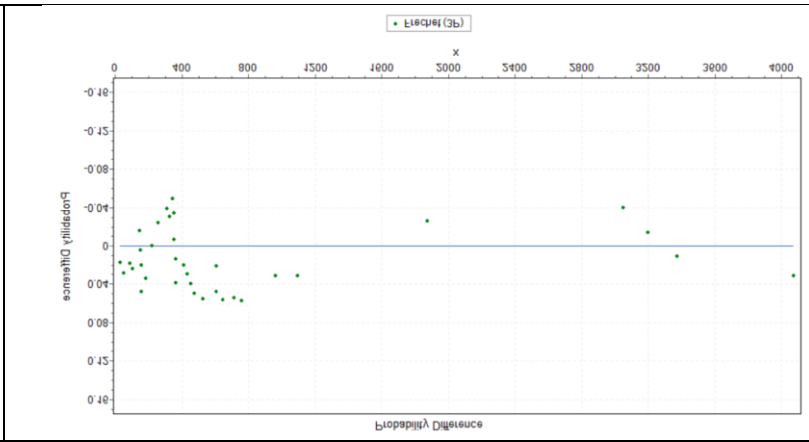
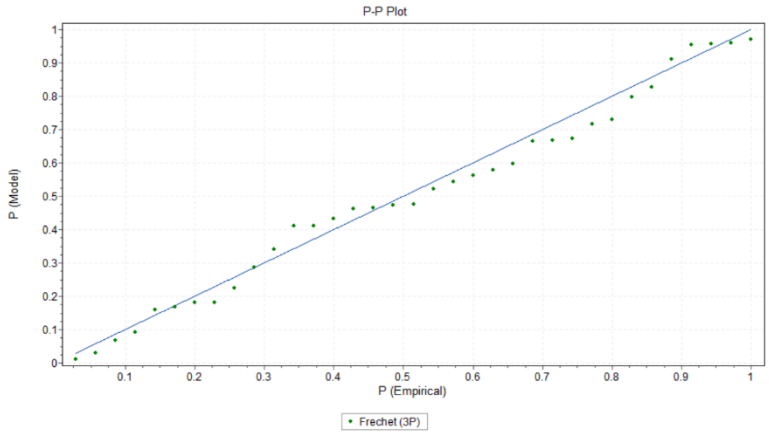
Heuristic Procedure	
DFD plot	
P-P plot	
Q-Q plot	

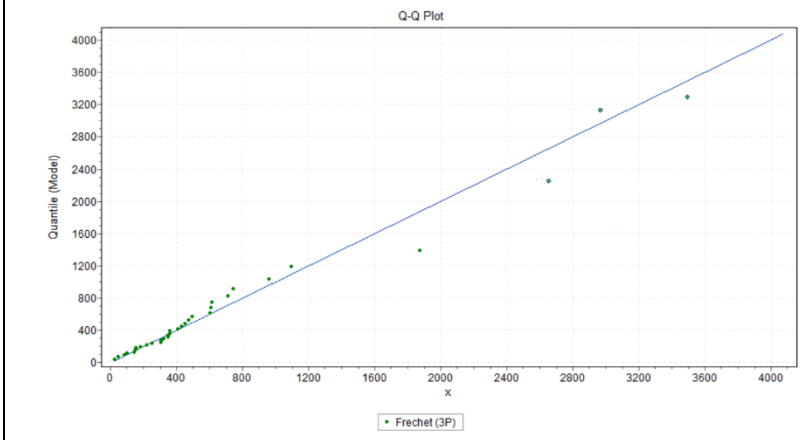
1) DFD plot : fitted distribution의 CDF와 input data를 기반으로 작성한 empirical distribution의 CDF의 차이가 0.05이상 크기 차이나는 지점이 거의 없다.

2) P-P plot : 분포 fitting의 중간 부분의 차이를 확인하기 위한 P-P plot을 확인한 결과 매우 큰 차이를 보이지 않았다.

3) Q-Q plot : 분포 fitting의 양 끝 부분의 차이를 확인하기 위한 Q-Q plot을 확인한 결과 매우 큰 차이를 보이지 않았다.

C. 3인 그룹 interarrival

<table>
<tr><th colspan="2">Heuristic Procedure</th></tr>
<tr><td>DFD plot</td><td>  </td></tr>
<tr><td>P-P plot</td><td> 
</td></tr>
</table>

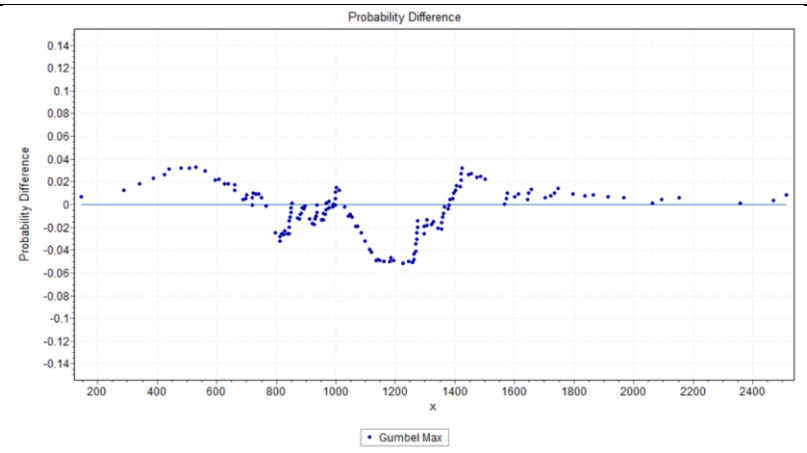
Q-Q plot	

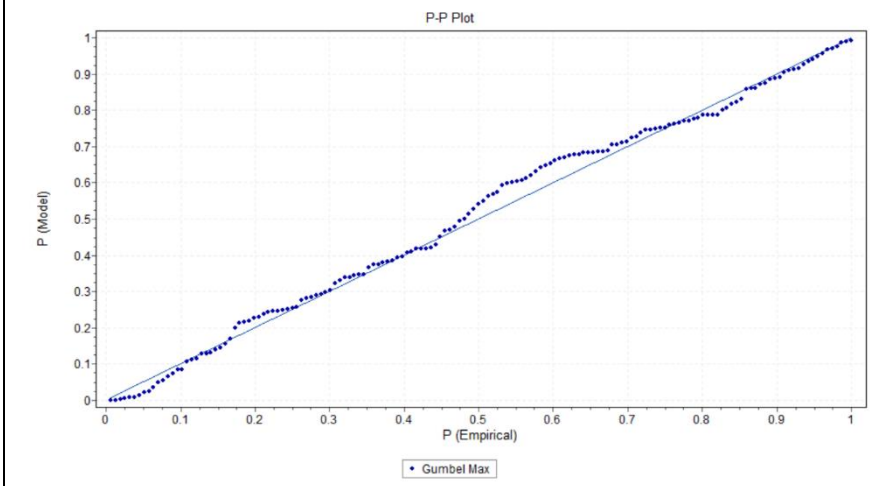
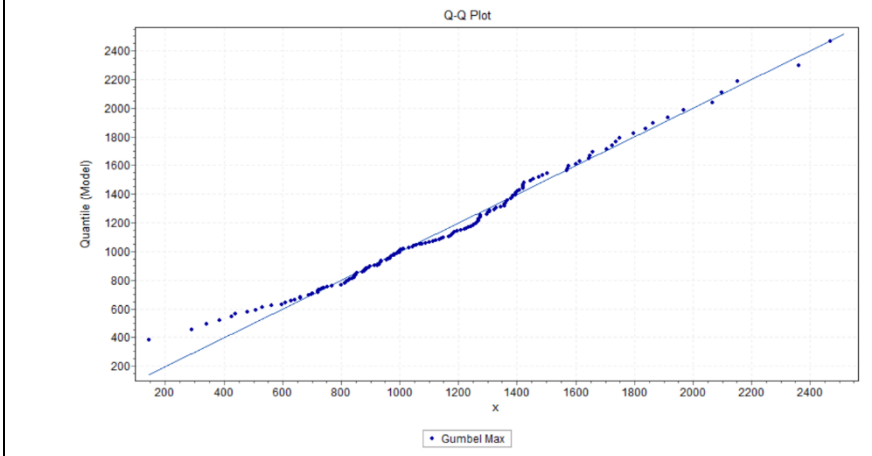
1) DFD plot : fitted distribution의 CDF와 input data를 기반으로 작성한 empirical distribution의 CDF의 차이가 0.05이상 크기 차이나는 지점이 거의 없다.

2) P-P plot : 분포 fitting의 중간 부분의 차이를 확인하기 위한 P-P plot을 확인한 결과 매우 큰 차이를 보이지 않았다.

3) Q-Q plot : 분포 fitting의 양 끝 부분의 차이를 확인하기 위한 Q-Q plot을 확인한 결과 매우 큰 차이를 보이지 않았다.

D. Service Time

Heuristic Procedure	
DFD plot	

P-P plot	
Q-Q plot	

1) DFD plot : fitted distribution의 CDF와 input data를 기반으로 작성한 empirical distribution의 CDF의 차이가 0.05이상 크기 차이나는 지점이 거의 없다.

2) P-P plot : 분포 fitting의 중간 부분의 차이를 확인하기 위한 P-P plot을 확인한 결과 매우 큰 차이를 보이지 않았다.

3) Q-Q plot : 분포 fitting의 양 끝 부분의 차이를 확인하기 위한 Q-Q plot을 확인한 결과 매우 큰 차이를 보이지 않았다.

결론

Heuristic Procedure를 통하여 모든 분포가 적절하게 추정되었음을 확인하였다.

② . Statistical hypothesis test

통계적으로 분포 추정이 잘 되었는지 확인하기 위하여 Chi-Square test와 Kolmogorov-Smirnov test(K-S test)를 이용하였다 귀무가설과 대립가설은 다음과 같다.

H0	The fitted distribution is correct
H1	The fitted distribution is not correct

	Interarrival			Service Time
	1인그룹	2인그룹	3인그룹	
p-value(K-S)	0.69653	0.86192	0.87166	0.44429
p-value(Chi)	0.9963	0.60799	0.99464	0.8746
Result	Fail to reject H0	Fail to reject H0	Fail to reject H0	Fail to reject H0

Chi-Square test 결과 유의수준 0.05에서 모두 귀무가설을 기각할 수 없었으므로 분포추정이 적합하게 되었다고 할 수 있다. 그러나 Chi-Squre test의 경우 표본의 수가 적은 경우 p-value가 높게 나오는 경향이 있다. 따라서 이를 보완하기 위하여 K-S 검정을 추가로 진행하였고 K-S 검정결과 유의수준 0.05에서 모두 귀무가설을 기각할 수 없다. 따라서 3개의 interarrival time과 1개의 service time 분포에 대하여 분포추정이 적절하게 되었다고 할 수 있다.

③. 결론

Heuristic Procedure과 Statistical hypothesis test를 이용한 총 5가지 방법을 통하여 분포 추정의 적합성을 확인한 결과 분포 추정에 문제가 없음을 확인하였다.
결과적으로 도출된 분포는 아래와 같다.

Input Data		분포	Estimator		
			Parameter1	Parameter2	Parameter3
Interarrival	1인 그룹	Weilbull(2p)	0.89876	501.58	
	2인 그룹	Log-Logistic(2p)	1.6152	163.13	
	3인 그룹	Frechet(3p)	1.4704	390.16	-112.21
Service Time		Gumbel Max	326.03	955.59	

III. Model Structure

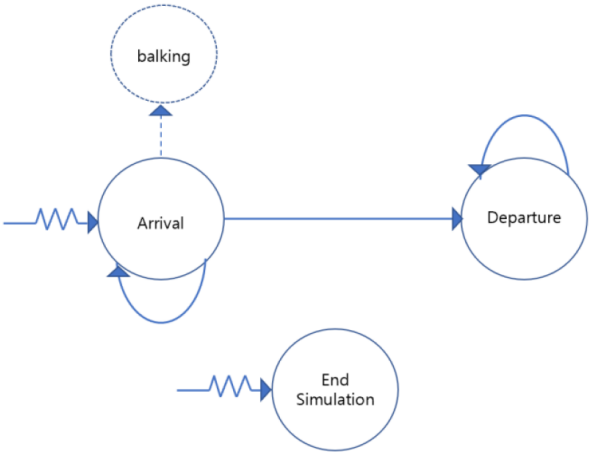
1. Model Assumption

시뮬레이션 모델의 가정은 다음과 같다.

1	Boundary	점심 시간 : 12시 00분 ~ 14시 00분
2	Initial Condition	시뮬레이션 실행 30분 이후부터 통계량을 업데이트한다. * 고른햇살에서 근무하시는 여사님께서 11시 30분부터 손님이 많이 오기 시작한다고 하여 11시 30분부터 시뮬레이션을 시작하고 12시부터 통계량을 업데이트했다.
3	고객 타입	1인 그룹, 2인 그룹, 3인 그룹의 손님만 있다고 가정한다. 데이터 수집 결과 4명으로 이루어진 그룹의 손님 수의 표본이 너무 적어 4인 그룹의 경우 3인 그룹의 일부로 편입시켰다. 각 그룹별 Service Time은 같은 분포를 가진다고 가정한다.
4	Server	현재 system 에는 4인용 테이블 6개가 있다. (Server의 개수=6) 4인용 테이블 하나를 줄일 때마다 2인용 테이블 2개를 늘려가며 Policy 1~6에 대해 시뮬레이션을 진행한다. ((0,6), (2,5), (4,4), (6,3), (8,2), (10,1)) 2인 테이블에는 1인, 2인 손님이 앉을 수 있으며, 4인 테이블에는 1인, 2인, 3인 손님이 앉을 수 있다. Service time은 최소 10분 최대 60분이다.
5	Queue	Queue1(2인 테이블 대기행렬), Queue2(4인 테이블 대기행렬)로 구분한다. (실제로는 하나의 Queue이다.)
6	Balking	손님이 도착하였을 경우 어떠한 테이블이 비어있지 않은 경우 Balking이 발생한다. 이때 고햇에서 대기하고 있는 손님이 6그룹 이상일 경우 balking은 무조건 발생하고, 손님이 4그룹 이상일 경우에는 50%의 확률로 balking이 발생한다.
7	Stopping Rule	시뮬레이션 실행 2시간 30분이 지나면 종료한다. (Initial condition을 위해 30분 소요)

2. Model Description

i. Event graph

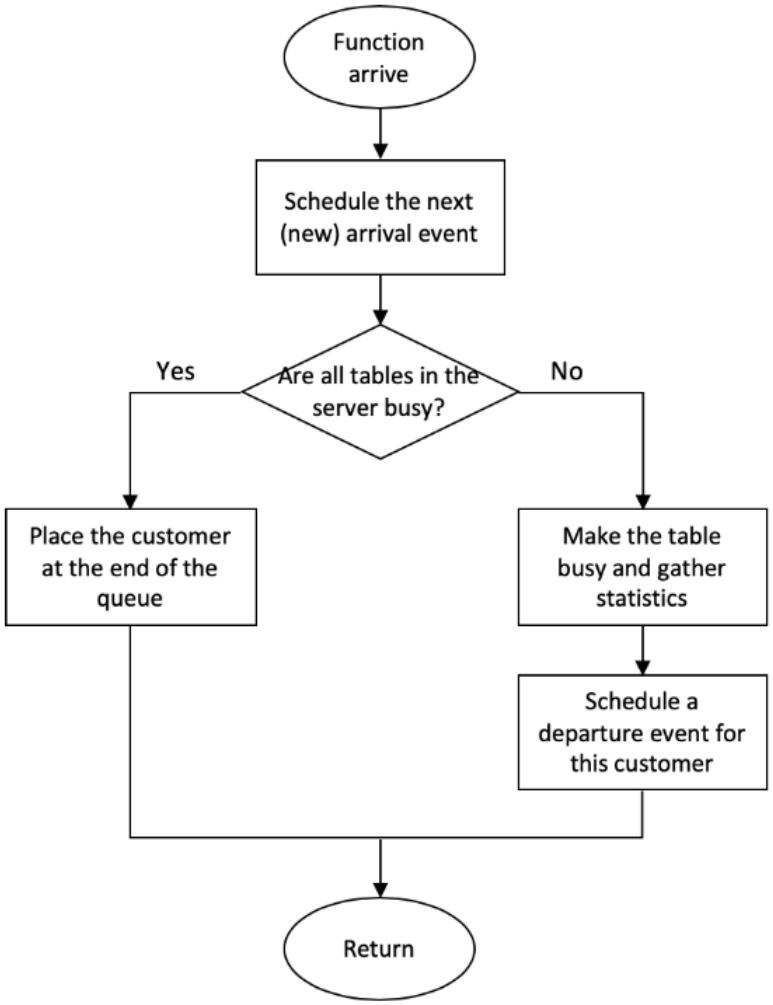


ii. Simulation model의 성분

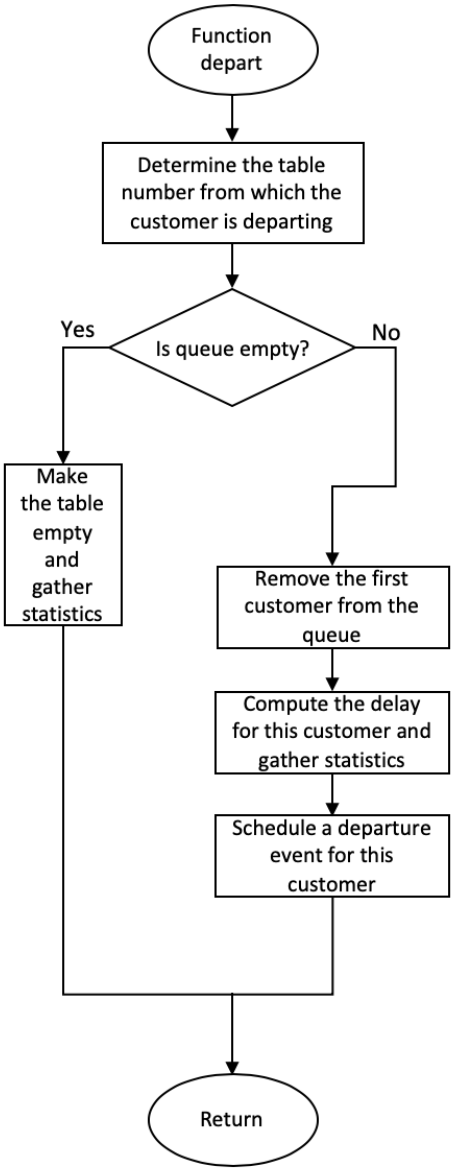
Entities			손님, table
Events			Arrival, Departure, Balking(Non-event function)
List	Event List	25	Event time, Event type [3]table number, [4]customer type for Departure Event
	Queue	1, 2	
	Table	3 ~	Dummy lists for utilization
State Variables	Sampst	Delays in Queue	Waiting time for each customer
		Service time	Service time for each customer
	Count variable	Serviced customers	Total Serviced customers and each group
		Balked customers	Total balked customers

3. Model Structure

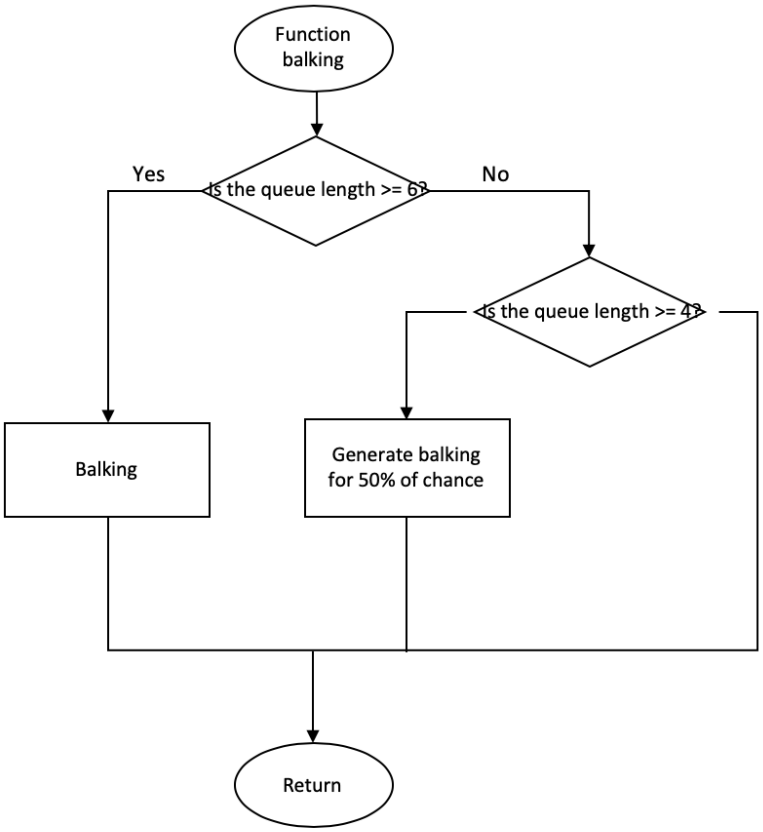
i. Arrival event flow chart



ii. Departure event flow chart



iii. Balking event flow chart



IV. First Simulation

1. First simulation

우리는 우리의 시뮬레이션을 두 번에 나누어 진행을 하였다. 첫번째 시뮬레이션을 통해 가상의 고른햇살에서 받을 수 있는 인원 수 (Capacity) 를 현재와 동일하게 24명으로 설정하여 4인 테이블을 하나씩 줄여 나가면서 고른햇살의 대기 시간에 어떠한 변화가 일어나는지 관찰하였다.

<Policy>

Policy 1	2인용 테이블 0개, 4인용 테이블 6개 (현재와 동일)
Policy 2	2인용 테이블 2개, 4인용 테이블 5개
Policy 3	2인용 테이블 4개, 4인용 테이블 4개
Policy 4	2인용 테이블 6개, 4인용 테이블 3개
Policy 5	2인용 테이블 8개, 4인용 테이블 2개
Policy 6	2인용 테이블 10개, 4인용 테이블 1개

2. Experimental results

<Phase1 simulation results>

Policy		Mean	S.D.	Ni	
1	(0,6)	10.5415	2.731079	32	12
2	(2,5)	7.4345	2.368365	24	4
3	(4,4)	4.8255	1.835192	21	1
4	(6,3)	3.963	2.266744	22	2
5	(8,2)	4.1015	2.5279	27	7
6	(10,1)	10.4335	4.880107	100	80

Phase 1의 시뮬레이션 결과 policy 4 와 policy 5가 손님들의 대기 시간이 가장 짧은 것을 확인할 수 있었다. 우리는 추가로 표본의 표준편차를 구하여 Phase2에서 실험을 몇 번을 더 추가로 진행할 것인지 확인하였다. Policy 6의 경우 기다리는 시간에 대한 측정치가 많이 퍼져 있어, 다른 Policy들에 비해 추가로 이루어져야 하는 실험이 많은 것을 확인할 수 있었고, 반면 Policy 3의 경우 표준편차가 다른 시뮬레이션에 비해 작아 추후 가중치를 주기 위해 단 한번의 실험만 추가로 진행하였다. (P*=0.90, n0=20, k=6, m=2, d=1)

<Phase2 simulation results>

Policy		waiting_time
1	(0,6)	8.386667
2	(2,5)	8.3725
3	(4,4)	6.91
4	(6,3)	3.955
5	(8,2)	2.695714
6	(10,1)	9.1365

Phase 2 시뮬레이션을 각각 추가로 해준 결과이다.

<Weighted Average of mean waiting time>

Policy		W1	W2	weighted
1	(0,6)	0.711497	0.288503	9.919825
2	(2,5)	0.896964	0.103036	7.531147
3	(4,4)	1.102929	-0.10293	4.610944
4	(6,3)	0.958778	0.041222	3.96267
5	(8,2)	0.796641	0.203359	3.81562
6	(10,1)	0.239932	0.760068	9.447692

3. Output Analysis

Phase2 시뮬레이션을 통해 가중치를 준 결과 역시 phase1과 동일하게 Policy 4와 Policy 5가 손님들에게 가장 짧은 waiting time을 주는 것을 알 수 있다. 위의 결과를 토대로 우리는 4인용 테이블을 2와 3으로 고정을 시키고, 2인용 테이블의 숫자에 변화를 주는 두번째 시뮬레이션을 진행하였다. 고른햇살의 협소한 공간을 생각을 하였을 때 받을 수 있는 손님의 인원 수 (capacity)를 동일하게 맞춘다고 하여도, 테이블의 개수가 늘어나게 되면 손님들이 식사를 하기 불편할 것이라고 생각하였기 때문이다.

V. Second Simulation

1. Second simulation

첫번째 시뮬레이션 결과 우리의 문제인식과 동일하게 3~4인의 그룹보다 1~2인으로 구성된 그룹이 훨씬 많이 오는 분식집의 특성에 맞지 않는 4인 테이블 개수라는 결론을 도출할 수 있었다. 우리의 두번째 시뮬레이션에서는 최종적으로 고른햇살에 현실적인 솔루션을 제안하기 위해서, 협소한 고른햇살에 2인 테이블과 4인 테이블을 어느정도 적절히 조합하는 것이 최적의 조합이 될 것인지 알아 볼 것이다.

<Policy>

Policy1	2인용 테이블 4개, 4인용 테이블 2개
Policy2	2인용 테이블 5개, 4인용 테이블 2개
Policy3	2인용 테이블 6개, 4인용 테이블 2개
Policy4	2인용 테이블 7개, 4인용 테이블 2개
Policy5	2인용 테이블 8개, 4인용 테이블 2개
Policy6	2인용 테이블 2개, 4인용 테이블 3개
Policy7	2인용 테이블 3개, 4인용 테이블 3개
Policy8	2인용 테이블 4개, 4인용 테이블 3개
Policy9	2인용 테이블 5개, 4인용 테이블 3개
Policy10	2인용 테이블 6개, 4인용 테이블 3개

2. Experimental results

<Phase1 simulation results>

Policy		Mean	S.D.	Ni	
1	(4,2)	12.1295	2.005365	21	1
2	(5,2)	7.8595	2.637117	21	1
3	(6,2)	5.468	2.546491	21	1
4	(7,2)	4.764	2.403314	21	1
5	(8,2)	4.671	2.442257	21	1
6	(2,3)	15.543	2.783226	23	3
7	(3,3)	8.6145	3.51729	37	17
8	(4,3)	7.067	2.83205	24	4
9	(5,3)	5.5345	1.739953	21	1
10	(6,3)	4.298	2.047686	21	1

Simulation 결과 Policy 3, 4, 5, 10 을 사용하였을 때 손님들의 대기시간이 가장 짧을 것이라고 예측이 되어진다. 첫번째 시뮬레이션때와 마찬가지로, Phase2 시뮬레이션에서 추가로 몇번의 실험을 진행해야 할지도 계산하였다.(P*=0.90, n0=20, k=10, m=4, d=1)

<Phase2 simulation results>

Policy		waiting_time
1	(4,2)	10.93
2	(5,2)	9.618889
3	(6,2)	5.374286
4	(7,2)	6.122
5	(8,2)	3.632
6	(2,3)	15.62538
7	(3,3)	11.56344
8	(4,3)	7.907143
9	(5,3)	5.34
10	(6,3)	4.11

Phase 2 simulation을 각각 추가로 해준 결과이다.

<Weighted Average of mean waiting time>

Policy		W1	W2	weighted
1	(4,2)	1.139159007	-0.139159007	12.29642123
2	(5,2)	0.984741668	0.015258332	7.886345339
3	(6,2)	1.018777418	-0.018777418	5.469759712
4	(7,2)	1.054918578	-0.054918578	4.689420571
5	(8,2)	1.045904077	-0.045904077	4.718694336
6	(2,3)	0.874755435	0.125244565	15.55331823
7	(3,3)	0.647770684	0.352229316	9.653202239
8	(4,3)	0.84485445	0.15514555	7.197344425
9	(5,3)	1.199831688	-0.199831688	5.573367263
10	(6,3)	1.130155223	-0.130155223	4.322469182

3. Output Analysis

Phase2 시뮬레이션을 통해 가중치를 준 결과 역시 phase1과 동일하게 Policy 3, 4, 5, 10이 손님들에게 가장 짧은 waiting time을 주는 것을 알 수 있다.

VI. Revenue Analysis

1. 목적

위의 분석에서 우리는 여러가지 대안들 중에서, 고객들의 대기시간을 가장 줄일 수 있는 정책 4가지를 제시하였다. 우리는 이러한 정책을 적용하였을 경우, 손님들의 편의 뿐만 아니라, 고른햇살의 매출에도 순영향을 미치는 것을 보여주기 위해 우리가 제시한 시스템과 현재 시스템의 비교 실험을 진행하였다. 이를 통해 고른햇살의 입장에서도 시스템의 변화를 통해 매출을 추가로 올릴 수 있는 기회가 될 것이다.

2. 분석

선행 시뮬레이션에서 output analysis 결과 효과적인 정책으로 선정된 (6,3)과 (8,2) 정책에 대하여 기존 정책인 (0,6)과 서비스 받은 총 인원에 유의미한 차이가 있는지 paired t-test로 검정하고자 한다.

Paired-t test의 경우 같은 seed number를 사용하여 input으로 인한 variance를 줄이고자 하였고, Welch test에서는 다른 seed number를 사용하여 독립성을 보장하였다.

	Paired-t Confidence Interval	Welch Confidence interval	결론
Policy(6,2)	(-14.907 , 5.797)	(-15.112 , 6.002)	0 포함 => 차이가 없다
Policy(7,2)	(-23.253 , -4.777)	(-23.381 , -4.649)	차이가 존재한다
Policy(8,2)	(-29.296 , -7.884)	(-27.948 , -9.232)	차이가 존재한다
Policy(6,3)	(-26.536 , -11.004)	(-29.686 , -7.854)	차이가 존재한다

3. 결론

paired-t test를 통한 통계적 검정을 통하여 기존 정책과 새로운 정책을 비교하였을 때 2인 테이블 7개와 8개, 4인테이블 2개로 운영하는 정책과 2인 테이블 6개, 4인 테이블 3개를 운영하는 정책이 더 많은 총 인원을 서비스할 수 있다는 것을 볼 수 있었다. 다시 말해 (8,2), (7,2), (6,3) 정책은 기존 정책보다 더 많은 손님을 동일한 점심시간 2시간동안 받을 수 있음을 의미한다. 따라서 새로운 정책은 고른햇살의 매출 상승에 기여할 수 있음을 확인하였다. 비단 그 뿐만 아니라, 정책 (6,2) 또한 의미를 갖는다. 정책 (6,2)로 테이블 수를 조정하여도 점심시간동안 서비스할 수 있는 총 고객의 수에 유의미한 차이가 존재하지 않는다. 매출이 줄어들지 않으면서 공간의 협소함을 해결하여 고객의 만족도를 더 높일 수 있는 계기가 될 것이다.

VII. Conclusion

본 프로젝트를 통해 고른햇살의 점심시간 고른햇살의 긴 대기행렬을 줄이는 것을 목적으로 진행되었다. 대기행렬을 줄이는 방법으로는 가게확장, 종업원수 충원 등 여러가지 방법이 있지만 확장과 충원의 방식은 많은 초기 투자비용을 요구한다. 우리는 최소한의 비용으로 시스템 개선효과를 내기 위하여 테이블의 재배치를 통한 개선안을 제시하였다. 우리는 고른햇살이 제공하는 분식 음식이 특성상 조리시간이 짧고 이를 필요로 하는 소비자들의 특성을 파악해 나가면서 이와 같은 개선안을 떠올릴 수 있었다.

우리는 테이블 재배치를 통한 개선정책이 효과적인지 검증하기 위해서 시뮬레이션을 시행하였으며 고른햇살 업주와 이용하는 손님의 입장에서 만족도를 측정하기 위하여 2가지 performance measure를 이용하였다. 먼저 손님의 입장에서 만족도를 높이기 위한 performance measure로써 waiting time을 기준으로 시뮬레이션을 한 결과 기존의 system보다 손님의 대기시간이 확연히 줄었음을 확인할 수 있었다. 두번째로는 고른햇살 업주의 매출상승에 따른 만족도의 performance measure로써 total number of serviced customers를 사용하였고 시뮬레이션 결과 고른햇살의 매출이 현행 system 보다 증가함을 확인하였다. 결론적으로, 테이블 재배치를 통한 system 개선 정책은 waiting time을 줄임으로써 고른햇살 이용객들의 만족도를 높이고 total number of serviced customers가 늘어나 매출이 증가하여 고른햇살 업주의 만족도를 높이는 win-win 정책이라고 할 수 있다.

안암은 여타 상권지역과 다르게 주요고객들이 고려대학교 학생으로 상인과 학생사이의 긴밀한 유대감정이 형성되어 있다고 할 수 있다. 예를 들어, 안암에 위치한 영철버거의 경우 영업 초기부터 꾸준히 학생들에게 저렴한 가격에 푸짐한 햄버거를 제공하여 많은 학생들을 배불리 해주었고, 영철버거가 학생들에게 제공한 추억은 후에 영철버거가 폐업위기에 몰리자 학생들의 모금운동으로 다시 회생하게 되는 계기가 되었다. 고른햇살도 마찬가지로 오랜 기간 안암에서 영업하면서 오직 영리만을 추구하는 식당이 아닌 학생들과 긴밀한 유대감을 가지는 식당이라 할 수 있다. 우리는 본 프로젝트가 고른햇살과 고려대학교 학생들이 오랜 기간 추억을 쌓으면서 학생과 상인사이의 상생관계를 돕기 위한 작은 기여가 될 수 있을 것이라고 생각한다.

VIII. How we built code?

시뮬레이션 코드는 기본적으로 C 언어를 기반으로 작성을 하였습니다. 수업 시간에 배운 Simlib 라이브러리를 기반으로 시뮬레이션 모델을 만들었습니다. 데이터 분석은 R 코드를 위주로 사용하였으며, R 코드로 제공이 되지 않는 경우 엑셀을 통해 직접 손으로 했습니다.

1. C 코드

<Simlib Library Variable 정의>

우선 simlib library에서 필요한 variable을 정의를 하였습니다. 숫자로 표현하는 것보다 따로 인식하기 쉬운 글로 표현을 하여 코드의 가독성을 높였습니다. Arrival 이벤트를 3개의 그룹으로 나누어 따로 발생을 시켰습니다. 1인 그룹, 2인 그룹, 3~4인 그룹의 분포를 따로 추정하였기 때문입니다. 또한 5번째 이벤트로 EVENT_STATISTICS를 정의하여, warmup 기간 동안의 데이터를 버리고 통계량을 새롭게 측정하도록 설정하였습니다. LIST는 QUEUE1 과 QUEUE2, 그리고 테이블 별로 리스트를 따로 정의를 하였습니다. 대기행렬을 두 개로 분할한 이유는 테이블의 종류를 2인용과 4인용으로 구분하였기 때문입니다. 하지만 이것은 어디까지나 가상으로 구분을 했을 뿐, 실제로는 하나의 대기행렬입니다.

```
1   /* External definitions for gohat. */
2   #include <stdio.h>
3   #include "simlib.h"             /* Required for use of simlib.c. */
4
5   #define EVENT_ARRIVAL1       1  /* Event type for arrival of group1 customer. */
6   #define EVENT_ARRIVAL2       2  /* Event type for arrival of group2 customer. */
7   #define EVENT_ARRIVAL3       3  /* Event type for arrival of group3 customer. */
8   #define EVENT_DEPARTURE      4  /* Event type for departure of a customer. */
9   #define EVENT_CLOSE_DOORS    5  /* Event type for closing doors at 2 P.M. */
10  #define START_STATISTICS     6  /* Event type for starting statistic */
11  #define LIST_QUEUE1          1  /* List number for group1,2 queue. */
12  #define LIST_QUEUE2          2  /* List number for group3 queue. */
13  #define SAMPST_DELAYS        1  /* sampst variable for delays in queue(s). */
14  #define SAMPST_SERVICES      2  /* sampst variable for service times */
15
```

<non-simlib variable, function 정의>

Arrival 이벤트, departure 이벤트를 표현해줄 함수 4개와 non-event function balking을 새롭게 정의 하였고, report 함수를 통해 최종 결과를 출력했고, init_model을 통해 모델의 초기화가 필요한 시점마다 적절히 초기화를 진행하였습니다.

```
16  /* Declare non-simlib global variables. */
17
18  int   num_tables1, num_tables2, serviced_customers, customer_type, num_balking,
19        RN_interarrival1,RN_interarrival2,RN_interarrival3,RN_service,RN_balking, serviced1, serviced2, serviced3;
20  float  length_doors_open, departure_time, warm_up;
21  FILE  *outfile;
22
23  /* Declare non-simlib functions. */
24
25  void arrive1(int RN_interarrival1, int RN_service, int RN_balking);
26  void arrive2(int RN_interarrival2, int RN_service, int RN_balking);
27  void arrive3(int RN_interarrival3, int RN_service, int RN_balking);
28  void depart(int list_table, int customer_type, int RN_service);
29  void report(void);
30  void balking(int customer_type, int RN_balking);
31  void init_model(void);
32
```

<Main 함수>

Main 함수를 시작하면서 가게가 얼마나 운영이 될 것인지 정의를 하고, 통계량을 언제부터 측정하기 시작할 것인지 정의를 해주고, 시뮬레이션을 진행을 할 때마다 수동으로 RN_XXX로 표현이 된 각 기능마다의 Random Number의 Seed를 새롭게 정의해 주었습니다.

```
main()  /* Main function. */
{
    int i, k;
    float start_statistic;

    /* parameters */
    length_doors_open = 2.5;
    start_statistic = 0.5;

    outfile = fopen("gohat_final_s.out", "w");
    fprintf(outfile, "Gohat closes after%16.3f hours\n", length_doors_open);
    fprintf(outfile,"\n       mean  mean    average");
    fprintf(outfile,"\n policy        waiting   service customer");
    fprintf(outfile,"\n(1~2,3~4)      time          time    in queue     served     balking     ");
    fprintf(outfile, "\n_____________________________________________________________________");

    RN_interarrival1 = 1;
    RN_interarrival2 = 2;
    RN_interarrival3 = 3;
    RN_service = 4;
    RN_balking = 5;
```

```
/* Open  and Write out files, report heading and input parameters.*/
fprintf(outfile, "\n stream %2d %2d %2d %2d %2d\n", i, RN_interarrival1, RN_interarrival2, RN_interarrival3, RN_service, RN_balking);

num_tables1 = 10;

for (num_tables2 = 1; num_tables2 <= 6 ; ++num_tables2){

    init_simlib();
    init_model();

    /* Set maxatr = max(maximum number of attributes per record, 4) */
    maxatr = 5;

    /* Schedule the first arrival. */
    event_schedule(generate1(RN_interarrival1), EVENT_ARRIVAL1);
    event_schedule(generate2(RN_interarrival2), EVENT_ARRIVAL2);
    event_schedule(generate3(RN_interarrival3), EVENT_ARRIVAL3);

    /* Schedule the gohat closing, and start Statistic*/
    event_schedule(60 * length_doors_open, EVENT_CLOSE_DOORS);
    event_schedule(60 * start_statistic, START_STATISTICS);
```

```
    /* Run the simulation while the event list is not empty. */
    while (list_size[LIST_EVENT] != 0) {

        /* Determine the next event. */
        timing();

        /* Invoke the appropriate event function. */
        switch (next_event_type) {
            case EVENT_ARRIVAL1:
                arrive1((int) RN_interarrival1, (int) RN_service, (int) RN_balking);
                break;
            case EVENT_ARRIVAL2:
                arrive2((int) RN_interarrival2, (int) RN_service, (int) RN_balking);
                break;
            case EVENT_ARRIVAL3:
                arrive3((int) RN_interarrival3, (int) RN_service, (int) RN_balking);
                break;
            case EVENT_DEPARTURE:
                depart((int) transfer[3], (int) transfer[4], (int) RN_service);  /* transfer[3] is table_list. */
                break;
            case EVENT_CLOSE_DOORS:
                event_cancel(EVENT_ARRIVAL1);
                event_cancel(EVENT_ARRIVAL2);
                event_cancel(EVENT_ARRIVAL3);
                break;
            case START_STATISTICS:
                init_model();
                sampst(0.0, 0);
                timest(0.0, 0);
        }
    }
    /* Report results for the simulation with num_tellers tellers. */
    report();
    num_tables1 -= 2;
```

<Init_Model 함수>

Init_model함수에서는 sampst, timest를 이용하지 않고 따로 통계량을 측정한 값들을 초기화 시키는 역할을 한다. 위의 main 함수를 보면 처음 시뮬레이션이 시작될 때, EVENT_STATISTICS를 할 때 사용되는 것을 알 수 있다.

```
115 void init_model(){
116
117     /* Set serviced_Customers = 0 */
118     serviced_customers = 0;
119     num_balking = 0;
120     serviced1 = 0;
121     serviced2 = 0;
122     serviced3 = 0;
123 }
```

<Arrival 함수>

EVENT_ARRIVAL1이 발생하여 arrive1 함수가 실행되면, 우선 첫번째로 다음 EVENT_ARRIVAL1을 생성시킨 후, 빈 테이블이 있는지 확인을 한다. 만약 빈 테이블이 있다면 바로 서비스를 받게 되는데, arrive1 함수와 arrive2 함수에서는 2인용, 4인용 테이블중 하나라도 비어 있다면 바로 서비스를 받을 수 있게 설정을 하였고, arrive3 함수에서는 4인용 테이블이 비어있는 경우에만 서비스를 받도록 설계하였다. 바로 서비스를 받게 되는 경우 EVENT_DEPARTURE를 생성하게 된다. EVENT_DEPARTURE를 발생시킬 때 transfer[3]에는 서비스를 받은 테이블의 번호(list번호)를 저장하게 되고, transfer[4]에는 customer_type(1인, 2인, 혹은 3인)을 받아서 EVENT_LIST의 attribute로 저장을 하게된다.

```
125 void arrive1(int RN_interarrival1, int RN_service, int RN_balking)  /* Event function for arrival of a customer to the bank. */
126 {
127     int table;
128
129     /* Schedule next arrival. */
130     event_schedule(sim_time + generate1(RN_interarrival1), EVENT_ARRIVAL1);
131
132     /* If a table is idle, start service on the arriving customer. */
133     for (table = 1; table <= num_tables1 + num_tables2; ++table) {
134         if (list_size[2 + table] == 0) {
135
136             /* This table is idle, so customer has delay of zero. */
137             sampst(0.0, SAMPST_DELAYS);
138
139             /* Make this table busy (attributes are irrelevant). */
140             list_file(FIRST, 2 + table);
141
142             /* Schedule a service completion. */
143             transfer[3] = 2 + table;  /* Define third attribute of type-two event-list record before event_schedule. */
144             transfer[4] = 1;
145             departure_time = sim_time + generate_service(RN_service);
146             sampst(departure_time - sim_time, SAMPST_SERVICES);
147             event_schedule(departure_time, EVENT_DEPARTURE);
148
149             /* Return control to the main function. */
150             return;
151         }
152     }
153
154     balking(1, RN_balking);
155 }
```

<Departure 함수>

EVENT_DEPARTURE이 발생하게 되면 departure함수가 실행되게 된다. 이때 위의 arrival 함수에서 받았던 테이블의 번호와 customer_type의 인자가 함께 들어온다. 주어진 Customer_type을 기반으로 customer_type 별로 서비스 받은 고객의 통계량을 측정하였다. 주어진 테이블 번호를 기반으로 다음 서비스 받을 고객을 결정하게 되는데, 만약 4인용 테이블에서 departure가 발생하였다면, queue1과 queue2를 모두 확인해야한다.

두 queue가 모두 비어있는 경우와 둘 중하나가 비어있는 경우에 기존의 방식과 동일하게 구현을 하였다. 하지만 두 queue에 모두 대기하고 있는 손님이 있다면 그 두 종류의 손님 중 누가 먼저 도착한 손님인지 구분을 할 필요가 있었다. 그렇기 때문에 simlib.c 파일에 list_check라는 함수를 따로 정의 하여 pointer로 LIST 별로 가장 처음과 끝의 데이터를 확인할 수 있도록 하였다. 그렇게 하여 두 종류의 손님 중 누가 먼저 왔는지 결정을 하고, 그 손님의 서비스를 시작을 하였다.

```
223  void depart(int list_table, int customer_type, int RN_service)  /* Departure event function. */
224  {
225      float time1, time2;
226
227      if (customer_type == 1){
228          serviced1 += 1;
229      }
230      if (customer_type == 2){
231          serviced2 += 1;
232      }
233      if (customer_type == 3){
234          serviced3 += 1;
235      }
236
237      serviced_customers += 1;
238
239      if (list_table > 2 + num_tables1) {
240          /* Check to see whether the queue is empty. */
241          if (list_size[2] == 0){
242              if (list_size[1] == 0){
243          /* all queues are empty, so make the table idle. */
244              list_remove(FIRST, list_table);
245              }
246              else {
247                  /* The queue is not empty, so start service on a customer. */
248                  list_remove(FIRST, 1);
249                  sampst(sim_time - transfer[1], SAMPST_DELAYS);
250                  transfer[3] = list_table;  /* Define before event_schedule. */
251                  transfer[4] = customer_type;
252                  departure_time = sim_time + generate_service(RN_service);
253                  sampst(departure_time - sim_time, SAMPST_SERVICES);
254                  event_schedule(departure_time, EVENT_DEPARTURE);
255              }
256          }
257          else {
258              if(list_size[1] == 0) { /*queue1은 비어있고, queue2만 있음*/
259                  /* The queue2 is not empty, so start service on a customer. */
260                  list_remove(FIRST, 2);
261                  sampst(sim_time - transfer[1], SAMPST_DELAYS);
262                  transfer[3] = list_table;  /* Define before event_schedule. */
263                  transfer[4] = customer_type;
264                  departure_time = sim_time + generate_service(RN_service);
265                  sampst(departure_time - sim_time, SAMPST_SERVICES);
266                  event_schedule(departure_time, EVENT_DEPARTURE);
267              }
268              else{ /*queue1도 있고, queue2도 있는 경우 => simlib에 새로운 함수 list_check 추가*/
269
270                  list_check(FIRST, 1);
271                  time1 = transfer[1];
272                  list_check(FIRST, 2);
273                  time2 = transfer[1];
274
275                  if(time1 < time2){
276                      list_remove(FIRST, 1);
277                      sampst(sim_time - transfer[1], SAMPST_DELAYS);
278                      transfer[3] = list_table;  /* Define before event_schedule. */
279                      transfer[4] = customer_type;
280                      departure_time = sim_time + generate_service(RN_service);
281                      sampst(departure_time - sim_time, SAMPST_SERVICES);
282                      event_schedule(departure_time, EVENT_DEPARTURE);
283                  }
```

```
                else{
                    list_remove(FIRST, 2);
                    sampst(sim_time - transfer[1], SAMPST_DELAYS);
                    transfer[3] = list_table;  /* Define before event_schedule. */
                    transfer[4] = customer_type;
                    departure_time = sim_time + generate_service(RN_service);
                    sampst(departure_time - sim_time, SAMPST_SERVICES);
                    event_schedule(departure_time, EVENT_DEPARTURE);
                }
            }
        }
    }
    else {
        /* Check to see whether the queue is empty. */
        if (list_size[1] == 0){
        /* The queue is empty, so make the table idle. */
            list_remove(FIRST, list_table);
        }
        else {
            /* The queue is not empty, so start service on a customer. */
            list_remove(FIRST, 1);
            sampst(sim_time - transfer[1], SAMPST_DELAYS);
            transfer[3] = list_table;  /* Define before event_schedule. */
            transfer[4] = customer_type;
            departure_time = sim_time + generate_service(RN_service);
            sampst(departure_time - sim_time, SAMPST_SERVICES);
            event_schedule(departure_time, EVENT_DEPARTURE);
        }
    }

}
```

<Balking함수>

Balking 함수에서는 처음 손님이 도착하였을 경우 어떠한 테이블이 비어있지 않은 경우 이후의 프로세스를 구현하였다. 이때 고햇에서 대기하고 있는 손님이 6그룹 이상일 경우 balking을 무조건 발생하도록 구현하였고, 손님이 4그룹 이상일 경우에는 50%의 확률로 balking을 하도록 구현하였다. 이때 lcgrand를 다시 한번 활용하여, 0과 1사이의 숫자를 랜덤하게 추출하여 0.5보다 작을 경우 balking을 발생시키는 식으로 구현하였다.

```
void balking(int customer_type, int RN_balking)  /* Event function for arrival of a customer to the bank. */
{
    if (list_size[1] + list_size[2] >= 6){
        num_balking += 1;
    }
    else {
        if(list_size[1] + list_size[2] >= 4 ){
            if(lcgrand(RN_balking) <= 0.5){
                num_balking += 1;
            }
            else{
                transfer[1] = sim_time;
                if (customer_type == 3){
                    list_file(LAST, 2);
                }
                else {
                    list_file(LAST, 1);
                }
            }
        }
        else{
            transfer[1] = sim_time;
            if (customer_type == 3){
                list_file(LAST, 2);
            }
            else {
                list_file(LAST, 1);
            }
        }
    }
    return ;
}
```

<Report 함수>

Report함수에서는 두 가상의 queue를 하나의 실제 queue로 합쳐서 통계량을 내기 위해서 simlib기본 내장 함수인 filest를 이용하여 각 queue의 평균 대기 손님을 구하고, 더하여 실제 queue의 평균적으로 시스템에서 대기를 하는 손님의 숫자를 구현하였다.

```
350  void report(void)  /* Report generator function. */
351  {
352      float mean_queue1, mean_queue2;
353      int max_queue1, min_queue1, max_queue2, min_queue2;
354      /* Compute and write out estimates of desired measures of performance. */
355      filest(1);
356      mean_queue1 = transfer[1];
357      filest(2);
358      mean_queue2 = transfer[1];
359      fprintf(outfile, "(%3d,%3d)%15.2f%15.2f%15.2f%15.2d%15.2d%15.2d%15.2d%15.2d\n",
360      num_tables1, num_tables2, sampst(SAMPST_DELAYS,-SAMPST_DELAYS), sampst(SAMPST_SERVICES,-SAMPST_SERVICES),
361      mean_queue1 + mean_queue2, serviced_customers, num_balking,
362      serviced1, serviced2, serviced3);
363  }
```

1-1. Changes in Simlib Code

<generate 함수>

lcgrand함수를 통해 난수를 추출하고, 1, 2, 3 그룹의 interarrival 분포 추정한 것을 inverse transform 시킨 함수에 넣어 각각 interarrival time을 발생시켰다. 서비스타임의 경우 최소 10분, 최대 60분의 가정을 하여, truncated 된 함수를 넣었다. 수업시간에 배운 V를 정의하여 inverse transform된 함수에 넣는 방식을 사용하여 구현하였다.

```
698  float generate1(int stream) /* 1인 그룹 => weibull distribution with lambda=501.58 k=0.89876 */
699  {
700      return (501.58 * pow(-log(lcgrand(stream)), 1.0/0.89876))/60.0;
701  }
702
703  float generate2(int stream) /* 2인 그룹 =>  2parameter log logistic distribution with alpha=1.6152 beta=163.13*/
704  {
705      return (163.13 * pow(lcgrand(stream)/(1-lcgrand(stream)), 1.0/1.6152))/60.0;
706  }
707
708  float generate3(int stream) /* 3인 그룹 => frechet distribution with alpha=1.4704 beta=390.16 gamma=-112.21*/
709  {
710      return (-112.21 + 390.16*(1.0/pow(-log(lcgrand(stream)), 1.0/1.4704)))/60.0;
711  }
712
713  float generate_service(int stream) /* service 분포 => gumbel max distribution with sigma=326.04 mu=955.59*/
714  {
715      float v,a,b;
716      a = exp(-exp(-(600-955.59)/326.03));
717      b = exp(-exp(-(3600-955.59)/326.03));
718      v = a + (b - a) * lcgrand(stream);
719      return (326.03 * (-log(-log(v))) + 955.59) / 60.0;
720  }
```

<List_Check 함수>

List_check함수는 기본적으로 주어지느 list_remove함수를 변형하여 구현하였다. 연결리스트 형식으로 구성된 list에서는 기본적으로 head pointer와 tail pointer를 가지고 있는 사실을 활용하여 간단히 구현하였다.

```
void list_check(int option, int list){

/* Check a record from list "list" and copy attributes into transfer.
   option = FIRST remove first record in the list
            LAST  remove last record in the list */
    struct master *row, *ihead, *itail;

    /* If the list value is improper, stop the simulation. */
    if(!((list >= 0) && (list <= MAX_LIST))) {
        printf("\nInvalid list %d for list_remove at time %f\n",
               list, sim_time);
        exit(1);
    }

    switch(option) {

        /* Remove the first record in the list. */
        case FIRST:
            row         = head[list];
            break;

        /* Remove the last record in the list. */

        case LAST:
            row         = tail[list];
            break;
    }

    /* Copy the data and free memory. */
    free((char *)transfer);
    transfer = (*row).value;
    free((char *)row);
}
```

1-2. Mersenne Twister RNG

우리는 추가로 LCGRAND가 갖는 치명적인 단점인 적은 시드 넘버를 해결하기 위해서 다른 난수생성기를 도입하려 하였다. 여러가지 찾아보던 중 Mersenne Twister Random Number Generator를 알게되어 도입을 하려 시도하였다. Mersenne Twister 난수생성기는 주기가 2^{19937}-1이며, 속도가 매우 빠르고, 난수의 품질이 매우 뛰어난 것으로 알려져있다. 이러한 장점에 끌려 Mersenne Twister코드를 인터넷에서 구해 도입하고자 하였고, 실제로 Mersenne Twister 난수 생성기의 난수를 이용하여 inverse transform시킨 interarrival time과 service time을 발생시키는 것까지 성공을 하였다. 하지만 simlib에 적용을 하려고 이식을 시켰는데 Random number를 새로 산출하지 않는 문제가 발생하였고, 다각도로 다양한 시도를 해봤지만 부족한 코딩실력에 발목이 잡히고 말았다.

```
MTRand seedRand(unsigned long seed) {
  MTRand rand;
  m_seedRand(&rand, seed);
  return rand;
}

/**
 * Generates a pseudo-randomly generated long.
 */
unsigned long genRandLong(MTRand* rand) {

  unsigned long y;
  static unsigned long mag[2] = {0x0, 0x9908b0df}; /* mag[x] = x * 0x9908b0df for x = 0,1 */
  if(rand->index >= STATE_VECTOR_LENGTH || rand->index < 0) {
    /* generate STATE_VECTOR_LENGTH words at a time */
    int kk;
    if(rand->index >= STATE_VECTOR_LENGTH+1 || rand->index < 0) {
      m_seedRand(rand, 4357);
    }
    for(kk=0; kk<STATE_VECTOR_LENGTH-STATE_VECTOR_M; kk++) {
      y = (rand->mt[kk] & UPPER_MASK) | (rand->mt[kk+1] & LOWER_MASK);
      rand->mt[kk] = rand->mt[kk+STATE_VECTOR_M] ^ (y >> 1) ^ mag[y & 0x1];
    }
    for(; kk<STATE_VECTOR_LENGTH-1; kk++) {
      y = (rand->mt[kk] & UPPER_MASK) | (rand->mt[kk+1] & LOWER_MASK);
      rand->mt[kk] = rand->mt[kk+(STATE_VECTOR_M-STATE_VECTOR_LENGTH)] ^ (y >> 1) ^ mag[y & 0x1];
    }
    y = (rand->mt[STATE_VECTOR_LENGTH-1] & UPPER_MASK) | (rand->mt[0] & LOWER_MASK);
    rand->mt[STATE_VECTOR_LENGTH-1] = rand->mt[STATE_VECTOR_M-1] ^ (y >> 1) ^ mag[y & 0x1];
    rand->index = 0;
  }
  y = rand->mt[rand->index++];
  y ^= (y >> 11);
  y ^= (y << 7) & TEMPERING_MASK_B;
  y ^= (y << 15) & TEMPERING_MASK_C;
  y ^= (y >> 18);
  return y;
}
```

```c
#include <stdio.h>
#include <stdlib.h>
#include "simlib.h"
#include "mtwister.h"
FILE  *outfile;
/* run this program using the console pauser or add your own getch, system("pause") or input loop */

int main() {
    outfile = fopen("test_generator.out", "w");
    int i;
    for(i=0; i<100; i++) {
        fprintf(outfile,"%f\n", generate1(2));
    }
    return 0;
    fclose(outfile);
}
```