프로젝트 최종보고서

공공데이터를 이용한 광역버스 시간표 자동생성 프로젝트

팀 원

20171266 박 건

20190544 배 상 원

120200199 김 종 현

목차

1. 프로젝트 개요

1.1 문제 제기 및 설계 목표

광역버스를 이용해 경기도에서 서울특별시로 이동하는 이용자 수는 매년 약 20만 명으로 많은 사람이 광역버스를 이용하고 있다. 다양한 업체에서 실시간 버스 위치 및 도착 예정 정보 제공 서비스를 운영하면서, 시민들의 편의를 증진시키고 있다. 다만, 아래와 같은 문제점이 있다.

1) 많은 정류장이 노선 시작점에 몰려 있어서 도착 예정 정보가 없다가 갑자기 도착 예정으로 표시되거나, 표시되기 전에 지나친다.

2) 배차 간격이 길어, 버스를 놓치면 오랜 시간 기다려야 한다.

3) 노선 시작점과 가까운 정류장을 버스가 정해진 시간에 도착하지만, 버스 운영 업체에서 시간표를 제공하지 않는다.

이를 위해, 경기도 버스정보시스템에서 제공하는 버스위치정보조회(REST) 서비스를 이용하여 축적한 버스 운행기록을 바탕으로 노선 별 운행 시간표를 구축한다. 위와 같은 광역버스뿐만 아니라, 대중교통 인프라가 부족한 지역은 지도 애플리케이션에서 제공하는 도착 예정 정보가 제대로 지원되지 않는다. 해당 경우도 각 자치단체의 공공데이터 서비스를 활용해, 버스 운행 시간표를 구축할 수 있도록 할 것이다.

1.2 현실적 제한조건

초기 프로젝트 목표에서는 여러 지역의 운행 데이터를 토대로, 기존 서비스의 사각지대를 해소시키고자 했으나, 각 자치단체에서 버스위치정보 공공데이터를 제공하지 않는 경우도 있고, 있더라도 실효성이 없는 경우도 있었다. 또한, 자치단체별로 제공되는 데이터 양식 및 형식(변수 명, 종류 등)이 통일되어 있지 않았다. 지역별로 다른 프로그램을 구성하는 것은 효율성이 떨어지고, 시간적 여유도 충분하지 않다고 판단했다. 따라서, 초기 목표와 달리 수도권 광역버스 중 주요 노선을 선택해 시간표를 완성할 것이다.

2. 요구사항

2.1 설계 목표 설정

경기도 버스정보시스템에서 제공하는 버스위치정보조회(REST) 서비스를 이용하여 축적한 버스 실제 운행기록을 바탕으로 노선 별 운행 시간표를 구축한다.

단계	세부 목표 및 진행
1	버스위치정보조회(REST) 서비스에서 노선 별 API 호출로 위치 정보 조회
2	조회한 데이터를 사용할 데이터 양식에 맞게 구문분석(Parsing) 및 저장
3	1~3 과정을 AWS서버에서 2주간 진행
4	노선 별 데이터를 읽어서 자료구조에 저장
5	시간표 생성 함수를 통해 시간표 작성

[표 1 – 세부 목표 및 진행 단계]

2.2 합성

구조체 설명

구조체	멤버 변수	설명
routeId_head	routeId_pointer next	Node pointer
	filepath_pointer down	Node pointer
	int routeId	노선 번호
	int totalStation	마지막 정류장 번호
filepath_node	filepath_pointer next	Node pointer
	char fileName[15]	출력 file 이름
	char outpathStr[34]	출력 path
_head_h	headh_pointer next_h	Node pointer
	headr_pointer down_h	Node pointer
_head_r	headr_pointer next_r	Node pointer
	headp_pointer down_r	Node pointer
	int maxStaSeq	최대 정류장 번호
	int routeId	노선 번호
_head_p	headp_pointer next_p	Node pointer
	headc_pointer down_p	Node pointer
	struct tm qdate	API 호출 시간
	int totalBus	총 버스 배차 수

	int totalStation	총 정류장 수
_head_c	headc_pointer next_c	Node pointer
	node_pointer down_c	Node pointer
	char plateNo[14]	차량 번호
	int busNoExp	호차 정보
	int stationExp	다음 정류장의 정류장 번호
bus_node	node_pointer next	Node pointer
	struct tm qtime	정류장 도착 시간
	int routeId	노선 번호
	int stationId	정류장 ID
	int stationSeq	정류장 번호
	int endBus	막차 여부
	int lowPlate	저상버스 여부
	char plateNo[14]	차량 번호
	int plateType	차량 종류 구분
	int remainSeat	남은 좌석 정보
_head_d	headd_pointer next	Node pointer
	dtt_pointer down	Node pointer
	int routeId	노선 번호
	int totalStation	총 정류장 수
	int totalBus	총 버스 배차 수
	int** flagTable	시간 별 API 호출 성공 여부 flag table
	int** sumTable	flag 합
dtt_node	dtt_pointer next	Node pointer
	int totalStation	마지막 정류장 번호
	int totalBus	총 버스 배차 수
	int** dailyTimeTable	일일 시간표

[표 2 - 구조체 설명]

2.3 분석

2.3.1 Time Complexity

함수이름	설명	시간복잡도
getRouteIdList	관심 있는 route의 수 : n 읽어오는 함수->O(n)	O(n)
getFolderPathList	폴더 읽는 함수	O(n)
getFilePathList	폴더 안의 일일 데이터 수 : m	O(m*n)

scanFile	while n:
	pushHeadR -> O(1)
	scanFolder
	while m:
	scanFile
	while(파일의 평균 정보 수: l) l:
	while(headC 에서 plateNo 확인):
	plateNo있는 유효한 노드 addNode
	기존에 없었던 plateNo를 새로 생성
		O(m*n*l)

makeAverage	while n:
	sumFunction1
	while m:
	sumFunction2
	sumFunction3
	for stationNum->i:
	for busNum->j:
	push node[단위]
	averageFunction
	while m:
	for stationNum->i:
	for busNum->j:
	평균계산(한 가지 경우를 조회함)
		O(n*m)

SUM	O(n)+O(n)+O(m*n)+O(m*n*l)+O(n*m)	
TOTAL	O(m*n*l)	

[표 3 – Time Complexity 계산]

2.3.2 초기 모델과의 차이점

초기 모델의 경우, 2차원 배열 구조에 장시간 같은 정류장을 지나고 있는 등(ex. 고속도로를 지나는 상황)의 필요하지 않은 정보도 모두 저장하여 많은 메모리가 필요하다. 이를 수정한 현재 모델은 stationExp(다음 정류소 번호로 기대되는 값)을 사용하여 필요한 정보인지 아닌지 여부를 판단할 수 있고, 판단 결과에 따라 list구조에 저장했다. 따라서 필요한 메모리도 크게 줄어들어 Space Complexity 효율성이 높아졌고, 접근성도 높아졌다. 다만, 데이터를 읽는 방식은 초기 모델과 다르지 않아서 Time Complexity는 동일하다.

2.4 제작

2.4.1 주요 함수 설명

단락	함수이름	설명
txt 파일 읽기	getRouteIdList	RouteId(버스 노선)을 가져오는 함수
		Input : -
		output : routeIdList
	GetFolder PathList	알맞은 Folder를 가져오는 함수
		Input : routeId_pointer routeIdList
		output : -
	getFilePathList	폴더 안의 일일 데이터 가져오는 함수
		Input : routeId_pointer routeIdList
		output : -
정보 추출하여 저장	scanFile	일일데이터의 정보를 추출하여 구조체에 저장하는 함수
		Input: FILE* inputFile, headp_pointer headP, node_pointer** dailyTimeTable
		output : -
추출한 정보 평균 계산	makeAverage	구조체의 저장된 함수를 바탕으로 평균을 내는 함수
		Input : routeId_pointer routeIdList
		output : -

[표 4 – 주요 함수 설명]

2.4.2 기본 알고리즘

1) 일일시간표

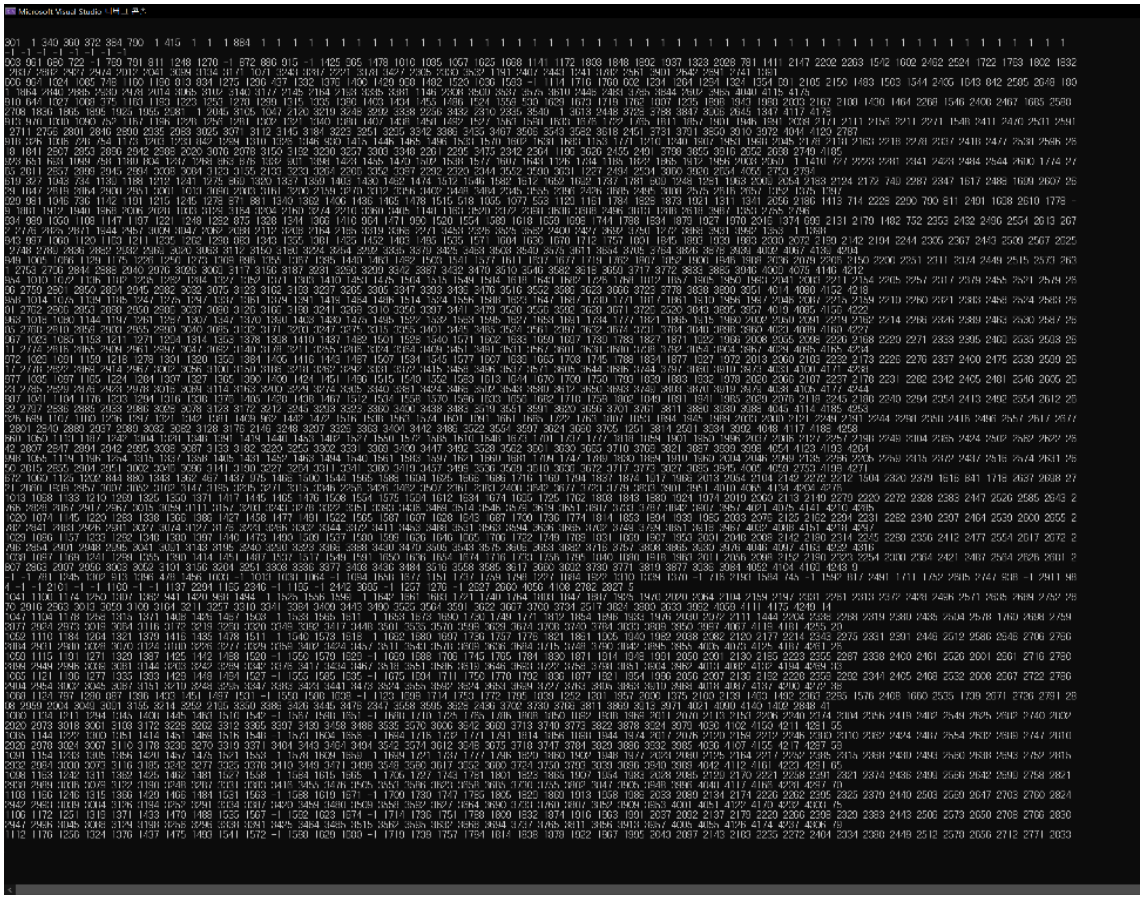
1. 데이터 파일에서 각 줄은 운행중인 버스 중 한 버스의 데이터이다.
2. 데이터가 옳바른 호차의 정보로 들어가야 하므로 plateNo를 비교한다.
3. 새로운 plateNo라면, 새로운 호차이므로 head_c를 추가한다.
4. 이미 입력된 plateNo라면, 해당 데이터의 필요 여부를 stationExp을 사용해 확인한다.
5. StationExp 값은 head_c에 있으며 bus_node가 새로 붙을 때마다 업데이트 된다.
6. (현 정류장 station number > stationExp) 다음 정류장으로 이동했다는 뜻이므로 새로운 노드로 list에 추가하고 stationExp값을 현 정류장 값으로 업데이트한다.
7. (현 정류장 station number = stationExp) 아직 다음 정류장으로 이동하지 못했다는 뜻이므로 해당 노드를 해제한다.
8. (현 정류장 station number < stationExp) 버스가 새로운 운행을 시작했다는 뜻이므로 새로운 head_c를 선언하고 이에 맞는 list를 만들어간다.

위의 알고리즘은 일반적인 경우이다. 여러가지 변수로 오류가 생기는 경우가 있는데, 이는 2.8에서 다룰 것이다.

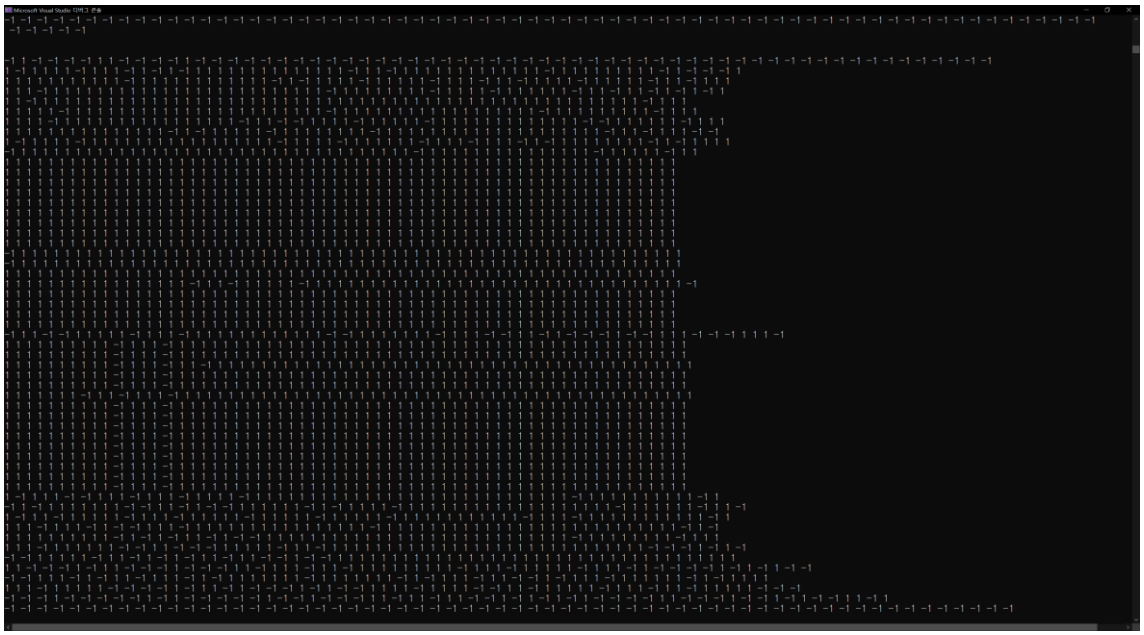
2) 일일 데이터의 평균 계산

1. 앞서 설명한 구조체들을 통해 확인하고 싶은 routeId에 접근한다.
2. 다시 각 날짜 별 데이터에 접근한다.
3. 접근한 날짜 별 데이터들을 다 더하여 평균을 계산할 준비를 한다.
4. 이 때 API 상 문제로 누락된 값들이 있는데, 이를 처리하기 위해 일일시간표와 같은 크기의 flagtable을 사용한다.
5. flag값은 데이터가 누락되지 않았다면 1, 누락되었다면 0이다.
6. 평균은 (Σ 데이터의 시간) / (Σ flag)로 구한다.
7. 이를 최종 시간표에 저장하여 txt파일로 출력한다.

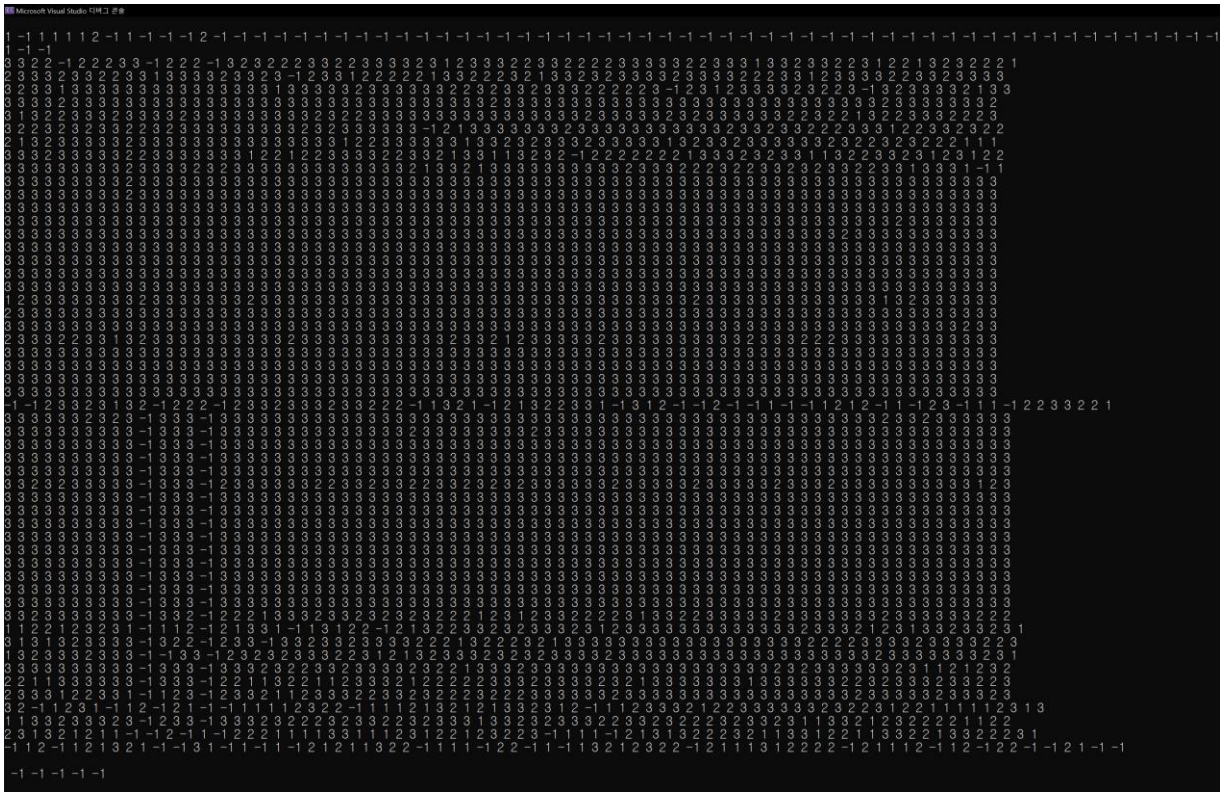
2.5 시험



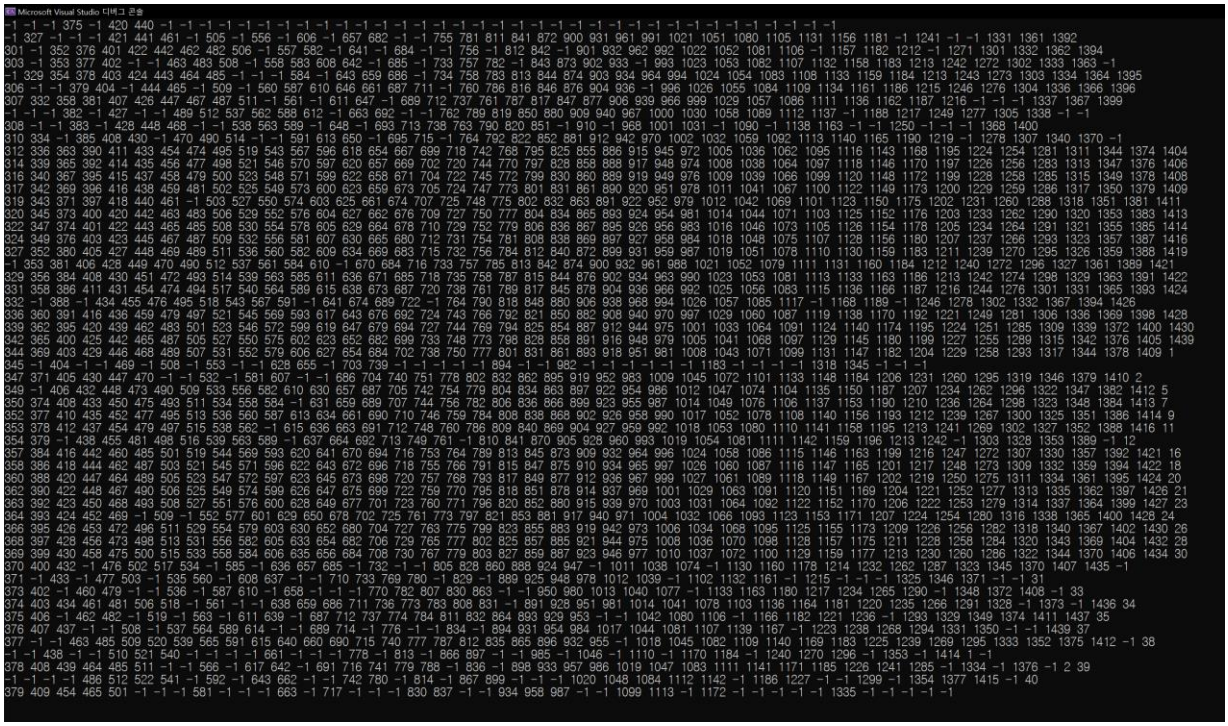
[사진 1 – daily time table]



[사진 2 – flag table]



[사진 3 – flag sum table]



[사진 4 – sum table]

2.6 평가

본 프로젝트는 통계에 실제 버스 운행기록을 기반한 시간표를 만들어 차고지에 가까운 정류장에서도 도착 정보를 조회할 수 있는 것에 큰 의의가 있다. 또한, 일반적인 경우만 고려하지 않고 종종 발생하는 데이터 오류에도 대응할 수 있도록 했다. 최종 출력도 초기에 기대했던 대로 얻을 수 있었다. 다만, 2.8에서 후술할 API 호출 문제는 완벽하게 해결하지 못한 아쉬움이 있다.

2.7 환경

프로토타입은 ssh 접속 프로그램을 통해 리눅스 서버에 접속하여 작업한다. 방대한 양의 데이터를 모두 전송해야 하는 문제점과 로컬 저장소에 접근하기 위해 사용한 라이브러리 문제로 이후 작업은 Microsoft Visual Studio 2019를 사용한다.

2.8 보건 및 안전

프로젝트에서 정의하고 있지 않은 입력이나, 읽어오는 데이터의 문제 등으로 인한 프로그램의 오작동이 없어야 한다. 또한 프로그램 동작 중에 일어날 수 있는 오류에 대처할 수 있어야 한다.

2.8.1 API 호출 문제

프로젝트에서 실시간 API 호출을 통해 데이터를 얻기 때문에, API 상 문제점이 그대로 나타나는 경우가 있는데 이는 아래와 같다.

1) GPS 인식 오류 등의 문제로 정류장 번호가 작아지는 경우
첫 번째 정류장을 1로 설정하고, 마지막 정류장까지 번호가 1씩 증가한다. 정상 운행을 하고 있다면 정류장 번호가 작아지는 경우는 없어야 한다. 그러나, GPS 인식 등의 문제로 잠시 작아졌다가 다시 다음 정류장으로 이동하면서 원래대로 돌아오는 경우가 있다 (EX. 정류장 번호 : 30 -> 16 -> 32). 이는 'diff' 라는 상수를 설정하여 해결했다. diff는 총 배차 수와 적당한 비율로 설정한다. 현재 정류장 번호와 diff를 더한 값이 예상 정류장 번호(현재 정류장 번호 + 1)보다 큰 경우만 호출된 데이터를 저장한다. diff를 적당히 큰 값으로 설정해주면, GPS 인식 오류를 해결할 수 있다.

2) API 호출 실패
알 수 없는 이유로 API 호출이 실패하는 경우가 종종 발생했다. 1분 간격으로 API 호출을 시도하는데 실패하는 경우 해당 시간의 데이터가 입력되지 않는다. 해당 부분은 최종 TimeTable에서 보면 빈 공간으로 처리된다. 앞 뒤 정류장 도착 시간으로 예상치를 계산하여 채워 넣는 방안도 검토했으나, 전체 호출 횟수와 비교했을 때 큰 영향을 주지 않을 것으로 판단했다.

3) 데이터 상 총 운행 버스 수와 하루 배차 수가 다른 경우

이미 마지막 정류장을 지나 차고지로 들어간 차량이 계속 호출이 되어 총 운행 버스 수가 원래 배차 수보다 크게 나타나는 경우를 발견할 수 있었다. 이는 프로그램이 종료되는 시간동안 인식이 된 경우이거나 운행을 종료한 차량의 GPS가 인식되어 새롭게 운행을 시작하는 차로 인식된 경우로 생각할 수 있다. 해당 오류는 데이터를 저장하는 단계에서 구분하기가 매우 어렵고, 항상 발생하는 오류도 아니기 때문에 조정이 까다롭다. 또한, 위의 원인이 아닌 다른 이유일 가능성도 있다. 이를 처리하는 방식에 따라 최종 시간표에 영향을 끼칠 수 있다.

2.8.2 통계 함수의 한계

일일시간표를 바탕으로 한 노선에서 배차 수가 같은 날들을 묶어서 각 그룹별로 평균치를 계산하여 최종 버스 운행 시간표를 만들었다. 이는 주중과 주말, 막차 시간과 운행하는 총 배차 수가 다르기 때문에 그로 인한 오차를 줄이기 위함이다. 그러나 연말, 연휴, 긴급 재난 등의 문제로 갑작스럽게 배차 수가 달라질 수 있다. 본래 시간표의 오차를 줄이기 위했던 방식이 오히려 오차를 만들 수 있는 가능성이 있는 것이다. 외부적 요인을 내부 프로그램의 수정으로 조정하기 어려운 점과 프로젝트 마무리 단계에서 새롭게 발견한 오류여서 한계점으로 남았다.

3. 기타

3.1 환경 구성

User	배상원
Model	HANSUNG TFX245
CPU	Intel Core i5-8265u 1.60HZ (Whiskey Lake)
RAM	16GB 2400MHz DDR4
GPU	Intel UHD Graphics 620 NVIDIA Geforce MX250
OS	Microsoft Windows 10 pro 10.0.18362
IDE	(C) CSPRO 리눅스 서버 (C) Microsoft Visual Studio Community 2019 16.7.5 (Python) Jupyter Notebook 6.0.3

User	박건
Model	레노버 아이디어패드 gaming 3i
CPU	Intel Core i5-10300H Processor (2.50GHz 8MB)
RAM	8.0GB DDR4-2933 DDR4 SODIMM 2933MHz
GPU	NVIDIA GeForce GTX 1650 4GB GDDR6
OS	Windows 10 Home 64
IDE	(C) Microsoft Visual Studio Community 2019 16.7.5

User	김종현
CPU	AMD Ryzen Threadripper 1900X
RAM	32GB 2400MHz DDR4
GPU	NVIDIA Quadro P2000 D5 5GB
OS	Microsoft Windows 10 Home 1909 18363.1198 (API) AWS Ubuntu Server 20.04 LTS (HVM)
IDE	(C) Microsoft Visual Studio Community 2019 16.0.30611.23 D16.7 (Python) Jupyterlab 2.2.9

3.2 팀 구성 및 수행기간

학번	이름	기여도(%)
20171266	박 건	**30**
20190544	배 상 원	**30**
120200199	김 종 현	**40**

프로젝트 수행기간 - 2020.10.28 ~ 2020.12.17 (50일)

3.3 Git-Hub 및 서비스 페이지

https://github.com/pilltong/Data-Structure-Project

https://kbtt.tistory.com/