TJHSST Hallway Traffic Simulation

Paul Woods
Pd5
10-31-08
Computer Systems Lab 2008-2009

Abstract

This paper details the creation of a traffic simulation specific to TJHSST. The goal is for the simulation to be accurate and modifiable using depth first search algorithms and traffic formulas and modeling techniques. The simulation will be mapped closely by printing out information and creating visuals. It will then be compared to real-life data and improved until close symmetry is reached. The simulation will then be used to discover and test real change possible for the simulation.

Introduction

With an increased influx of students each year, the hallways in TJHSST are becoming crowding and, at times, hard to navigate. The simulation will make it easier to view hall traffic and find a way to provide traffic relief to areas with high traffic.

The research will be done through collection of real-life data, applying logic and data collected to the simulation, changing variables in the simulation, and monitoring model change versus the real-life data. The goal is to create an accurate model of the school and use the model to discover real life improvements that could be made.

Before creating the experiment, background research involving similar experiments in the past and experiments that can be applied to this particular experiment were viewed.

Background

One such source was *Continuum Crowds, which was created* by Adrian Trueille, Seth Cooper, Zoran Popovic in conjunction with the University of Washington, Electronic Arts. The project is designed to determine how to realistically model large crowd movement without collision detection. Collisions and movement were calculated together--as opposed to separately--because crowds tend to already know which areas are likely to be congested and adjust *before* reaching the congestion.

They created several simulations, including one with 24 people in a hallway, a 2000 person army retreating from a 8001 person army, and a 16 square city block. The researchers concluded that the crowds actually moved more smoothly and realistically when calculating the collisions and movement together, as opposed to separately. The researchers mentioned future areas for continued research which included tighter packed areas, areas where people do not have a common goal in movement, and what they described as "posse chasing," in which certain people are avoided and others are looked for.

Another such source is *Finding Multi-Constrained Feasible Paths By Using Depth-First Search, carried out by* Zhenjiang L and J.J. Garcia-Luna-Aceves The focus of the research was developing a depth first search algorithm that operated with an unusual number of restrictions. The algorithm was specifically designed for developing routing systems, which requires the consideration of several constraints (such as bandwidth, reliability, end-to-end delay, jitter, and cost).

Past experiments specificic to TJHSST have also been done, such as Alex Kotkova's *Traffic Dynamics in Scholastic Environments.* The program works on finding a way to map traffic in an TJHSST-esque environment.

Development Sections

The program was tested by implementing the proposed algorithm and recording the time required by the algorithm to find and select routing pathways. The results were that the algorithm can work very effectively, but works best with a smaller number of possible locations.

The input the program will receive currently consists of students with a randomized schedule, and a hash containing a small-scale model room and hallway layout as shown by **Figure 1.** To test the program in its current state, I ran a simulation in which students would start in one area and be required to find their way to the designated path. This preliminary simulation assumed that each room was approximately three minutes apart and students would leave their classes at exactly 10:05 and would remain in their goal destination once they arrived. I then output student data and then combined it to get the data shown in **Table 1.**

The program is being made primarily in the C programming language, though it is possible other languages (such as C++ and Java) will be used later in the project to compensate for the weaknesses of the C programming language.

To test the program in its current state, I ran a simulation in which students would start positioned in a room and attempt to find their way to their destination: a different room. This preliminary simulation assumed the following:

1) Each room is located approximately three minutes apart from each other.
2) Students will attempt to take the quickest possible path to their destination.
3) Students will stay in their destination once they arrive.
4) Students move at the same speed, regardless of traffic conditions.
5) All students left their initial location at 10:05 and tried to make sure they arrived at their next location by 10:15.

The initial input into the program is as follows:

- Data containing the miniature school map, written directly into the program code* in the form of a hash.
- Randomly created student schedules

*At this point, there was no point in inputting data by reading a file because the map size manageable. The time and processing used to open and read an input file did not make such a method useful.

Results, Discussion, Conclusion, and Recommendations

The results of the experiment were very positive. Every student was able to locate his next period and arrive before the start of the next class (10:05). However, by looking at the data, it was clear that the hallways were not utilized efficiently. For instance, at 10:11, the Hall5 hallway had six times as much traffic as Hall4.

The results of the experiment were very positive. Every student was not only able to locate his or her destination, but he or she was able to arrive before there before the start of the next class (10:15).

However, there were some problems. Looking at the data, it was clear that the hallways were not utilized efficiently. For instance, at 10:11, the traffic through Hall5 hallway was 600% as congestion as the traffic through Hall4.

This may have been a result of the fact that students did not take into account traffic when selecting movement. Also, the order of the schools could have been switched so to alleviate the traffic.

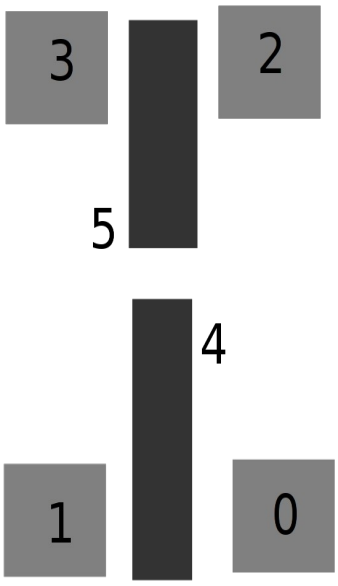
In the future, when the program is expanded to include the entire school, the program will likely be testing by collecting actual student movement data. There are several ways to do this, including counting the number of total students passing through certain hallways between classes, attaching tracking devices to willing participants, or asking a group of students to respond to survey questions about where they traveled and which areas they consider to be congested.

For the second quarter, I plan on experimenting with the first program to find a way to make hall traffic more uniform in the small simulation. Then, I plan on expanding the input given to the program to include the entire school and the entire student body. To allow the program to process this data, I will need to change the way the program calculates student movement by implementing some sort of search algorithm to

determine the pathway. I've been researching the a-star search and depth first algorithms, and I will likely incorporate these when designing the algorithm. Hopefully, at that point, the program will be able to output traffic data for the entire school, and I will be able to test it by comparing it to actual student movement during the day.

Appendices:

Figure 1:



Hall Traffic Given Time:

Hall	10:08	10:11
4	60%	10%
5	40%	60%

Example Output From The Program:

StudentNameStart 10:08 10:11 Finish

Student0 Area3 Area5 Area2 Area2
Student1 Area1 Area4 Area5 Area3
Student2 Area1 Area4 Area5 Area3
Student3 Area2 Area5 Area4 Area0
Student4 Area1 Area4 Area5 Area2
Student5 Area3 Area5 Area2 Area2
...

Code:

```
#include <stdio.h>

#include <math.h>

#include <stdlib.h>

#include <time.h>

#include <string.h>


int maxRooms=6;

int maxPaths=4;

int numStudents, spacePerStudent, maxTime; //the maximum number of rooms and max number of rooms connected to each room


void writehash(int* hash, char *ofilename1)

{

	FILE* outfile;

	outfile= fopen(ofilename1, "wb");
```

```
        fwrite(hash,sizeof(int),sizeof(int)*maxRooms*maxPaths,outfile);
        fclose(outfile);
}

void readhash(char* ofilename1)
{
        maxRooms=6;
        maxPaths=4;
        long lsize;
        printf("works\n");
        FILE* outfile;
        int *temphash;

        outfile= fopen(ofilename1, "rb");



        fseek(outfile,0,SEEK_END);
        lsize=ftell(outfile);
        rewind(outfile);
```

```
	temphash=(int*)malloc(sizeof(int)*lsize);


	fread(temphash,sizeof(int),lsize,outfile);
	fclose(outfile);
	free(temphash);


	int i;
	for(i=0; i<24; i++)
		printf("hash[%i]=%i\n", i, temphash[i]);
}


int main(int argc, char* argv[])
{
char *ofilename1="hashdata7.txt";


	numStudents=10;
	maxTime=4;
	spacePerStudent=2;  //first slot is current location, second slot is next location
	int *numStudentsRoom; //stores the number of students in each room;  index is room number
	numStudentsRoom=malloc(sizeof(int)*maxRooms);
	int *hash; //stores path between rooms and hallways, index is hallway
```

hash=malloc(sizeof(int)*maxRooms*maxPaths); //index of room indicates paths connected to it; hash*maxPaths gives room

//value of -1 indicates no path

int *students;

students=malloc(sizeof(int)*numStudents*spacePerStudent);

/*

demo info:

0: room1

1: room2

2: room3

3: room4

4: hall1

5: hall2

***/**

hash[0]=4;

hash[1]=-1;

hash[2]=-1;

hash[3]=-1;

hash[4]=4;

```
hash[5]=-1;

hash[6]=-1;

hash[7]=-1;

hash[8]=5;

hash[9]=-1;

hash[10]=-1;

hash[11]=-1;

hash[12]=5;

hash[13]=-1;

hash[14]=-1;

hash[15]=-1;

hash[16]=0;

hash[17]=1;

hash[18]=5;

hash[19]=-1;

hash[20]=2;

hash[21]=3;

hash[22]=4;

hash[23]=-1;


writehash(hash, ofilename1);

readhash(ofilename1);
```

```
/*
for(i=0; i<24; i++)
printf("hash[%i]=%i\n", i, hash[i]);
*/

int value;

for(i=0; i<numStudents*spacePerStudent; i=i+spacePerStudent)
{
	value=rand()%(maxRooms-2); //leave out hallways for this example
	students[i]=value;
	while(value==students[i])
	{
		value=rand()%(maxRooms-2); //leave out hallways for this example
	}

	students[i+1]=value;
}

//for(i=0; i<numStudents*spacePerStudent; i++) //creates student and designates schedule
//printf("student[%i]=%i\n", i, students[i]);
```

```
//Following code will determine how the students travel


int *paths;

paths=malloc(sizeof(int)*numStudents*maxTime); //array stores the paths the students are taking.  Index is student and then the steps at the time intervals

int i3;

int beststep; //best next step

int nextstep; //next step to take to get to finish

int step; //stores last step

int start, end;

int *possiblePaths;

int possible;

int t; //stores time

int flag;




        for(i=0; i<numStudents*spacePerStudent; i=i++) //designate path for student to take, i is index of student

        {

        flag=0;

        //need to set the path to th nextstep NEXT CLASS NEXT THING TO DO

        nextstep=students[i*spacePerStudent];

        //paths[i]=nextstep; //first time interval has all students at starting location

        end=students[i*spacePerStudent+1];
```

```
step = students[i*spacePerStudent];
paths[i*maxTime]=students[i*spacePerStudent];

for(t=1; t<maxTime; t++)
{



start=step;
beststep=-1;
        for(i3=0; i3<maxPaths; i3++)  //finds possible paths from start to finish
        {
        if(flag==1)
        {
        beststep=-1;
        break;
        }
//      printf("i:%i,%i step:%i\n", i, i3, step);

                possible=hash[step*maxPaths+i3];
        //      printf("possible:%i beststep:%i\n", possible, beststep);

                if(possible!=students[i*spacePerStudent])
```

```
                {
                if(possible==-1)
                        break;

                if(possiblePaths[i3]==end)
                        {
                        beststep=possible;
                        break;
                        }
                if(possible>beststep && beststep!=end) //for this model, greatest value
is always best unless actual value
                        beststep=possible;
                }

        //      printf("possible:%i beststep:%i\n", possible, beststep);
                }
        paths[i*maxTime+t]=beststep;
        step=paths[i*maxTime+t];
        if(beststep==end)
        {
        flag=1;
        }
        }
```

```
        }

for(i=0; i<numStudents*maxTime; i++)

{

//printf("paths[%i]=%i\n", i, paths[i]);

if(paths[i]==-1)

paths[i]=paths[i-1];

}


//PRINT OUT THE STUDENT ARRAYS


printf("\nStudentName\tStart\t10:08\t10:11\tFinish\n\n", i/numStudents, paths[i],
paths[i+1],paths[i+2],paths[i+3]);


int room41;

int room42;
```

```
int room51;

int room52;



room41=room42=room51=room52=0;



for(i=0; i<numStudents*maxTime; i=i+maxTime)

{

        printf("Student%i\tArea%i\tArea%i\tArea%i\tArea%i\n", i/maxTime, paths[i],
paths[i+1],paths[i+2],paths[i+3]);

        if(paths[i+1]==4)

                room41++;



        if(paths[i+2]==4)

                room42++;



        if(paths[i+1]==5)

                room51++;



        if(paths[i+2]==5)

                room52++;

}



printf("\nHall\t\t\t10:08\t10:11\n");
```

```
printf("\n4\t\t\t%i0%\t%i0%", room41, room42);

printf("\n5\t\t\t%i0%\t%i0%\n", room51, room52);




//array=malloc(sizeof(int)*200);


//printf("%i", sizeof(int));


}
```

Sources

Traffic Dynamics in Scholastic Environments by Alex Kotkova

Continuum Crowds by Adrian Trueille, Seth Cooper, Zoran Popovi

Finding Multi-Constrained Feasible Paths By Using Depth-First Search by Zhenjiang L and J.J. Garcia-Luna-Aceves