CS290 Final Report

Intelligent Geographic Transportation Systems implemented in OpenTTD

Charlie Croom and Cameron Muesco

Advisor: Drew McDermott
Abstract

For this project we designed and implemented an Artificial Intelligence program that plays Open Transport Tycoon Deluxe (OpenTTD or TTD), a business and construction simulation game. In OpenTTD, players attempt to earn profits by transporting passengers and goods using roads, rails, airplanes, and boats. Players are given a random map populated with cities, industries, natural resources, and obstacles, such as trees and water. They must create a transportation network connecting the sources of passengers (cities) and goods (industries and natural resources) in the most efficient and profitable way possible. In addition, they must maintain this network and deal with in-game events such as vehicle aging and breakdown, technology upgrades, offered subsidies, and actions by competitors operating on the same map.

Our goal was to develop an AI for OpenTTD that created an efficient transportation network that was both profitable in the game and theoretically viable in the real world. The AI attempts to design and implement a theoretically efficient network while working within and adapting to the constraints of the OpenTTD environment (ex. construction costs, terrain features, subsidies, and competitor actions). Through the development of this AI we hope to gain a better understanding of the modeling and design of transportation networks and other similar systems.

1. Introduction

The AI developed for this project, officially named ZooElite, consists of two major levels of operation. The ‘lower’ level of operation performs tasks that allow for the tile-level, physical construction of transportation routes between destinations. It determines and optimizes the placement and construction of new bus and rail stations, finds and builds routes between these stations, and creates vehicles to service routes. This area of ZooElite was the primary responsibility of Charlie Croom and is detailed in the first part of this paper.

The second, ‘higher’ level of the AI was the primary responsibility of Cameron Musco and is detailed in the second part of the paper. This level of the AI determines which routes to construct and which order to construct them in. It also schedules and coordinates the completion of various in-game tasks, including station construction, route construction, vehicle purchase, vehicle updates, and financial actions, such as loan obtainment and repayment.

This first section of the paper will describe the design constraints and final implementation of the tile-level construction features of ZooElite. In this section we will cover the algorithms and methods used to handle the construction tasks assigned to the “lower-level” of the AI by the route planner. In addition, we’ll discuss the successes and shortcomings of the AI in these areas, as well as the applications to real-world networks. Part one of this paper was completed by Charlie Croom and is written from his perspective.
2. OpenTTD, CargoDest and NoAI

2.1. OpenTTD and Our Goals

In the past 15 years since Chris Sawyer originally released Transport Tycoon, a number of independent developers have worked to re-code the entire game as an open-source project. Only during this past semester was the game finally released without any dependencies on the original game files. One has to wonder why so much effort was put into reviving a game which was merely a precursor to the more commercially successful “RollerCoaster Tycoon” series. The answer is that Transport Tycoon is so simple. The focus of the game is purely on building transportation networks and the game doesn’t concern itself with stock, advertising or advanced 3-d features which force micro-management onto the user.

This means that the game has virtually unlimited flexibility as a macro-level modeling tool, especially since it has been open-sourced. Many of the more experienced players are better at the game only because they are able to create more efficient networks through a combination of network planning, track routing and rail signal and junction creation. We aimed to create an algorithm which would create extremely efficient networks and simultaneously model real-world networks.

Of course, OpenTTD isn’t the real-world, and as such, not every factor can be accounted for. The game has an engine and built-in mechanics which we chose not to modify. The objective was not to create an AI which generated the most revenue in the game, but rather an AI which created a viable real-world network in the game. Our stipulations for a “viable real-world” AI were fairly straightforward:

1. Always have a viable network. In terms of graph theory, we wanted to limit the number of connected components we created. While our AI will occasionally create two connected components, it only does so if there is a compelling financial reason to create a new component (I.E. two very large, very close cities or a lack of large cities around the current connected component)
2. Create a financially stable AI. This ended up being harder than anticipated. In the real world, a large part of the cost of developing a network is buying land and laying the initial rail. Our main costs stemmed from bus building costs, a flaw which will be addressed later in this paper. Despite this, our AI managed to stay out of the red and continued to expand over time.
3. Connect as many cities as possible and provide as many transportation options as possible for our passengers.

Given these goals we set out to see how we could develop an algorithm and plan which would simultaneously accomplish all of the above. While OpenTTD also supports industries, we chose to focus on the passenger side of things as towns are far more interesting than industries in the game.
2.2 CargoDest

Put simply, the original Transport Tycoon game was stupid. Train signals were ineffective, stations were severely limited in size, airports had a size cap similar to that of the New Haven airport and passengers had no explicit destination. Perhaps an apt phrase would be, “If you come, they will go.” In the real world this would be equivalent to every single passenger deciding, “I just want to go somewhere today”, showing up at the station, and hopping on the first bus/plane/train/boat. This leads to some interesting game mechanics, namely the most financially successful business plan is to connect the two largest and most distant cities. Obviously this isn’t how the real-world works (Imagine if everyone from New York was willing to go to China instead of Boston), but we were stuck with these mechanics until CargoDest came along.

Developed as a branch of OpenTTD, the CargoDest mod is still actively in development and assigns each passenger a destination at the outset of their trip. Although it’s not finished yet, the mod aims to better model real-world passenger flow. This means that it will generate symmetric demand for passengers between cities (Modeling return trips). As the mod functions today, passengers are assigned a destination as a function of destination city size and distance (modeling commuting, inter-city movement and so on).

Most importantly the mod aims to scale passenger generation as a function network size. Right now, the mod takes the number of passengers generated by a city to be fixed (I.E. Out of its control) and simply allocates this fixed number of passengers to the available destinations offered by transport companies. The aim, though, is to control the number of passengers a city generates based on how many places those passengers could go. So the city might generate 50 passengers with 1 available destination in the network, but 250 with 4. This model simulates globalization and the increase in travel as networks mature and cities become less distant in terms of travel time. Unfortunately, at time of writing, this feature was non-functional, see later in the paper for more on how this affected our AI.

2.3 NoAI Framework

The NoAI framework separates the game’s engine from the AIs, meaning AIs may only access game information through method calls and may not modify any game settings or game mechanics. The framework is written in Squirrel, a language similar to Objective C. Neither Cam nor I had any prior experience with the language, which meant a lot of time spent reading the Squirrel API. In the end, the main data objects in squirrel are arrays and tables, which are essentially associative arrays. We found Squirrel to be an adequate language which was able to accomplish nearly everything we asked it to. Beyond that, our coding was only limited by the information available to us through the NoAI API.
3. Approach

3.1 Overall Structure

While we knew that implementing our ideas would have to be separated into two separate parts, Cam and I worked closely to decide from our experience what the best approach to creating our network would be. Both real-world economics and game mechanics suggest that trains are the most efficient mode of transportation because they cost less than airplanes but possess larger capacities. In the game, passengers pay only based on how far and how fast they traveled, so players are not concerned with setting prices, nor do airplanes net higher fares (other than that they are faster).

Given this, however, rail stations are also the most difficult to build because they require a significant amount of oddly shaped space to allow for track convergence. In contrast, bus stations are very simple, requiring only a single tile. Based on this information and our experience with real-world, big-city networks, we developed a two-tiered approach. The map would first be divided into regions based on the sizes and distances of various cities to one another; each region would contain a single train station. This train station would have bus stations connected to it, and additional bus stations would also be built in each city within the region.

So the first-tier would be intra-regional. This tier would consist of busses running from a single city to the train station which would serve as the hub for the region. Passengers traveling within the region would have a single transfer at the train station where they could go to any other city also in the region.

The second-tier would be inter-regional. This tier would serve as our main source of revenue and main means of moving people by relying on the aforementioned efficiency of trains. Once people were brought to the station by busses, trains would shuttle them to any other region in our network where they would again be picked up by busses and taken to their final destination.

This method, we felt, simulated the difficulties real-world transportation companies currently face. Namely, there is no way to build a new high-capacity station in a location central to a town. This can be seen in large cities with high-capacity airports like Hong Kong, Detroit and in almost all mid-sized cities either because these high-capacity terminals are too large to fit close to the city or because the city doesn’t provide enough PAX throughput on its own.

If you are interested in the algorithms used to determine regions and their centers, you can look at part 2 of this paper which goes more in-depth into the subject; my area of focus was in building the physical stations and determining placement at the micro-level which we’ll turn to now.

3.2 Intra-Regional Building

As we defined our regions in the previous section, we know that they must be geographically close (otherwise they would be in a different region). This means that intra-regional transportation alone will not be very profitable since the stations will be fairly close together. In the game, back-tracking actually incurs a cost rather than a profit, so it’s possible that some of these routes will actually
be losing money if they must travel back in the direction the trains came from. Thus the main point of these routes is not to transport between cities, but rather to link cities to the regional stations so that trains may transport people further away. Thus even though busses may incur a minor loss due to backtracking, the route is still profitable because those passengers pay much more to take a long-distance train ride.

So our primary focus in developing a region is maximizing PAX flow from cities to stations. The first step in this process is creating a large PAX pool by covering as much of the city as possible with bus stations. Each bus station has a coverage radius of about 3 tiles in each direction, meaning that only passengers from buildings up to 3 tiles away will come to the station. Thus to generate the most passengers, we’d ideally cover the entire city.

The bus station placement algorithm is designed to find the optimal placement for n-more stations in a given city taking into account any pre-existing stations. First, we obtain a set of tiles which covers the entire city by estimating the size of the city based on the number of buildings in a city. Next we run a series of tests on each tile in this set to determine if the placement is potentially viable. The requirements for such a placement are that the tile is either a road tile (Drive-through station) or is clear and next to a road tile (Depot type station), in addition we remove any station placements which would cause coverage overlap with existing stations.

With the list of valid placements, we determine the amount of passengers each placement individually would generate and store that in the list. Finally, we run a combinatorial algorithm which generates a list of unique and valid n-station arrangements, meaning that these new stations coverage radii wouldn’t overlap. Now we can easily find the absolute best configuration of n-new stations.

While a combinatorial algorithm is horribly inefficient in theory, steps were taken to ensure that the station placement was completed instantly. First, we made the assumption that the center of the city (Called the seed tile) is the densest region. Thus, if we place our first station as close to the center as possible, we get both a very high demand station and we cut down the complexity of future calculations. In addition, we can make the assumption that our AI will start close to the beginning of the game, meaning that cities will either be medium sized and contain no previous stations, or will be large, but already contain stations. By making this assumption, we again cut down the number of possible bus station placements to consider. Utilizing these assumptions, it will be very uncommon for us to ever add more than 5 stations at a time which means instant generation of the optimal station positioning.
Next we move on to the issue of building and routing busses within a town. Assume we know where the regional station is and where all the stations in the town are. First, we must determine how many busses should be built. This is done by looking at three things, how many passengers total the stations generated last month, how far the busses must travel, and how many people the busses transported in the previous month. Using this data we can figure out what percent of the passengers were transported last month. If this number is sufficiently high (above 60%), we are happy. Otherwise, we need to create more busses to get this figure higher (~80%).

Based on the number of passengers we failed to transport and the distance busses must travel, we can develop a fairly simple equation which estimates how many more busses we’d need to satiate demand. After testing, this method showed some weaknesses, namely it was impossible for any number of busses to ever satiate the total demand (especially without competition) in a large city, thus a saturation point was added to prevent busses from clogging the streets.

In order to route busses most efficiently, a greedy solution to the Travelling Salesman Problem was chosen. Obviously, the shorter the buses have to travel, the faster we can push PAX to the regional station. However given the easily implementable choices (Greedy or combinatorial), the Nearest Neighbor (NN) greedy algorithm was chosen to save speed. Starting at the regional station and using the NN algorithm yields nearly instant routing. This algorithm also yields a total distance of $1.25 \times \text{perfect_shortest_distance}$ on average, and given the negligible impact of traveling an extra few tiles for busses, this is acceptable.

Repeating this same process on every city in a region will yield a completely connected intra-regional network. If the region consists of more than one city, it will often be somewhat profitable, allowing us to generate minor income while we gear up for the big rail routing task ahead.
3.2 Inter-Regional Building

Rail transport is the primary source of revenue for ZooElite. As such, it was extremely important to make it as efficient and scalable as possible. There are a few different aspects of building these regional stations to explain, the first of which is the actual station construction. Assume for now, that we’ve located the free red plot of land as shown below (We’ll discuss this more later). On this plot, we need to do two things, build the rail station (yellow zone), and then hookup bus stations (Blue zone) to it which connect to the cities in the region.

The rail station builder is capable of building both vertically and horizontally oriented stations, either as a terminus (as above) or as a pass-through station with any number of platforms. The station design was done by hand and then converted into an algorithm. Because of the way the stations are designed, it makes them scalable to any number of platforms and platform lengths requested. Once we have the station built, we can build bus stations and hook them up to towns. Although the physical building of stations isn’t that interesting, it was rather time-consuming to ensure that all the signals and track worked in every configuration.

What’s more interesting is the method by which we choose the plot to build on (The red square in the above picture). When Cam’s part of the AI decides that it needs to build a station, it calls a method and passes the ideal location and size. We take that location to be the center of our search and create a list of tiles in a large radius around that location. Next, we assume that these tiles uniquely define a plot of land for which they would be the top left corner, effectively yielding a list of plots.

The AI then eliminates all plots of land which A) Are not clear or buildable, B) cannot be made flat across the entire plot, or C) would have blocked entrances to the station (I.E. If the purple arrow in Figure 2 above were blocked by something). By ensuring that the tiles in front of the track exit are clear,
we raise our chances of a successful pathfind when connecting tracks to the station. In the case of pass-
through stations, the AI can attempt to change which side of the plot the tracks converge to in order to
avoid conflicts.

Now that all the invalid placements are eliminated, we score each plot based on three values,
proximity to the given location, cost to level the plot, and proximity to a city. These three metrics are
multiplied by constants defined in constants.nut which allows the AI to easily switch what properties it
emphasizes. In OpenTTD, it is often better to build closer to cities even if it costs more because the cities
will grow around the station, thus eliminating the need for additional bus stations in that area (since
people can walk directly to the train station). From this point all that’s left to do is pick the highest
scoring plot and try to build on it. If something fails, we can abort and simply try the next best plot.

Now ZooElite has produced rail stations for each region and bus stations within the region. All
that’s left to do is connect them. Cam crossed over to help handle this aspect of the AI, which we
borrowed and modified from another AI, trAlns. That AI has a very code-heavy double-track pathfinding
program which we felt would be quicker to port that to implement ourselves. Double tracking is a crucial
part of our AI as it allows trains to travel back and forth quickly in both directions between stations.

![Figure 3 - A large regional station. Notice the brown area towards the top where the plot was leveled.](image)

4. Strengths of ZooElite

There are many very unique things ZooElite is capable of doing which no other AI is capable of. In
addition, some things it just does better or more efficiently. What follows is a discussion of some of the
major successes we had in programming ZooElite
4.1. Networking

It goes without saying that from the very start, ZooElite was intended to be a network based AI. As outlined in the goals section, ZooElite’s primary objective was to create a very nice looking network which connected as many cities as possible. Below is an in-progress game with only ZooElite’s assets shown. Cities can be seen as the black figures which are roads, each light blue rectangle is a rail station, and each small, light blue dot in the cities are bus stations. At this point in the game, ZooElite has created two connected components which it will eventually connect.

Figure 4 - A network map from ZooElite

ZooElite is the only AI we know of which implements a clustering algorithm to generate regions (note that each station services 1 – 4 cities in Figure 4). The other important feature of ZooElite networks are junctions, this is where one set of tracks merges into another set. See part two of this paper for a more in-depth discussion of network creation.

4.2 Station placement and creation

From an algorithm stand-point, it’s hard to beat ZooElite at bus station placement. Its methods always generate the best possible station configuration because all possibilities are considered. In terms of rail stations, many other AIs are pre-programmed with fixed station sizes and configurations (I.E. always 2 platforms). ZooElite is capable of creating any size station in a variety of configurations assuming a large enough plot of land exists. In addition, it possesses a scoring function which allows the user/developer to tweak station placements to favor cost, distance, or town proximity. This functionality is extremely valuable when creating larger regional transfer stations with high throughput, which might overwhelm other AIs.

4.3 Route Management

By utilizing object oriented programming, ZooElite is capable of creating logical routes in addition to physical paths. Each logical route consists of one contiguous track path and is uniquely
defined by a junction. Thus each time the route builder creates a junction in the tracks, a new route must be created. If there is no junction on a new path, however, then a pre-existing route can simply be extended. Below is the same map as network map as above, but with routes color coded on.

Figure 5 – Color coded routes on a network. Right: Potential extension of routes as paths expand

Note that while routes may share the same physical track in some areas, each route has its own set of trains which only run on that line. By doing this, we reduce the capital needed to build new routes. Unlike other AIs which will create a new set of trains for each city pair, our AI is capable of extending a route to service a new station once it’s connected to the network. Note in the diagram below that once physically connected, the red line will presumably be extended to service the new stations as shown.

This method not only leads to reduced capital on route expansions, but also helps to reduce track congestion and general clutter on the network. This means once it gets going, ZooElite is capable of expanding very rapidly.

5. Discussion of shortcomings and areas for further work

Despite the amount of time involved with coding ZooElite, there are still a number of areas where ZooElite is inferior other AIs. This section is intended to provide some insight into ZooElite’s shortcomings.

5.1 Financial Stability

If ZooElite were a company, it would be receiving federal bail-out money for risky investments. In hindsight, there is no reason for ZooElite to struggle with finances as much as it does. The reasoning behind this weakness is a two-fold problem of poor coding and game mechanics.
The first issue is the way the game works. Companies receive income based on how far PAX travel from their destination thus the income function is basically linear. Assuming we have PAX moving between two cities on a route for which we need two engines to satiate demand, the main cost associated with building this route is the cost of those two engines. Compared to this cost, the variable cost of running the trains is negligible. Therefore, if we create a route that moves people a distance of 40 tiles, this is 1/3 as profitable as a route that moves people 120 tiles.

Many other AIs exploit this economy by making their first train line as long as possible, that is by finding two large and very distant cities and connecting them. However, this method isn’t compatible with our graph theory network model nor does it make real-world sense. The real world equivalent would be connecting Boston and D.C., but skipping over New York initially. Therefore, ZooElite suffers from a large startup cost. Because the first two regions it connects are usually close, the profits for the route are occasionally barely enough to overcome the cost of building the route and vehicles, thus it can take a long time to gain enough capital to rapidly expand. We attempted to remedy this situation by placing a minimum length constraint on our first couple routes – however this comes at the cost of long term network efficiency.

The second cause of financial instability is poor programming. Because of the division of labor required by the CS 290 guidelines, Cam worked on route and large-scale planning and I worked on tile-level implementation. The plan was that he would handle a financial loop, however during testing we found that we often ran out of cash in the middle of atomic operations (Building track, building vehicles, etc.) The only solution for this was to add a waiting loop in various parts of my code which would wait for cash if it ran below a certain amount ($50,000). This is very sub-optimal. Ideally my methods would know the cost of each operation before attempting it, however this type of estimation fell in between our two areas and didn’t receive enough attention. Thus we had to implement a simple loop which will not allow any operations until our bank balance reaches an acceptable level. With more time we’d ideally implement a better cost estimation system which would help adaptively manage company assets.

5.2 Vehicle Management

During testing, we also discovered that our buses were not able to meet demand for travel in large cities. This causes a bottleneck at the source of PAX, and we wanted to avoid such a bottleneck at all costs. Unfortunately, that’s exactly what the problem was, costs. We quickly discovered that it was hard to build as many busses as my portion of the AI was requesting, since it sucked up a large amount of funds without any immediate ROI. Thus we settled for a lower number of initial busses, which ZooElite can adjust later in the game.

This too proved difficult, however, due to the functions available to ZooElite. Because the CargoDest mod isn’t part of trunk, NoAI support is not available for it. Thus the only figures available for measuring transport rates are percent of passengers transported from a city and number of passengers waiting at a station. In terms of busses, we can reliably use percent transported as a performance indicator since PAX demand is roughly symmetric, however trains are almost impossible to gauge.
We can, however, get the number of passengers waiting at each regional station on a route and try to minimize that through the addition of trains, but the problem is that we cannot tell where the passengers at regional stations are waiting to go. If they are waiting to go via bus into the city, ZooElite will build trains forever to try to fulfill what it thinks is rail demand. Thus the best approach was to take the numbers waiting at each rail station and analyze the deviation. If there was a high deviation, it was largely the case that PAX were bottlenecking waiting to go into a single city, but if the deviation was low, it might be the case that inter-regional transport is bottlenecking is occurring and more trains should be created.

In the end, ZooElite suffers from a lack of information. One potential fix for this would be to take a few cycles to analyze the current loads of all vehicles on the route. If we checked everyday for a month to see if vehicles were full or empty, we’d have an accurate idea of demand. The downside to this is that A) It’s impossible to compute this initially when no vehicles exist and B) We must wait for a vehicle to fully cycle its route, a time during which we can’t do anything else. It’s worth experimentation in the future, but for now we’ve chosen to stick with the original method.

5.3. City-Regional Links

When ZooElite’s overall plan was developed, it was known that busses would probably be a bottleneck for large cities, however we felt that busses were the best way to simulate real world networks and also the best way to push passengers out to regional stations. Once we began running ZooElite against other AIs, we quickly found ZooElite’s financial performance to be lackluster. Another train-based AI quickly bested us using fairly similar methods to our own, but by exploiting a particular game mechanic.

This “feature” of the game is linked stations. Originally, bus and train stations had to be immediately adjacent to be part of the same station, meaning people could transfer from busses to trains within the station and so on. However, with the open-sourcing came a crucial change. Stations can now be “linked” up to 10 or 15 tiles away. Other AIs exploit this by building rail stations on the outskirts.
of cities (like ZooElite does), but then filling the cities with bus stations, which are linked to the rail stations. This means that the rail station and all bus stations in the city act as the same station and people from across the city can magically teleport to the train station. Effectively, other AIs get the same PAX production ZooElite does without using any busses, which saves them a lot of capital. We feel that our AI is more realistic in this regard, but nevertheless are considering branching our code to mimic this behavior, thereby increasing financial success.

Getting PAX to go directly to the regional station eliminates a transfer and allows direct connections to high-capacity routes. In the end, this is the most important skill in a human player, which leads to economic success. In this regard AIs will never be as good as humans, for instance, above is a picture of one of my personal games. The stations are perfectly wedged in the middle of two cities, which creates massive PAX supply. While an extremely elaborate AI could figure out all the tile terraforming that would need to occur, it’s simply very difficult for anyone but a human to be able to build such customized stations such as these. Thus it’s hard to compete with humans. Instead ZooElite aims to compete with other AIs and to create a very interesting network.

6. Lessons Learned and Intellectual Value

This venture into the world of AIs was propelled by an interest not only in AIs, but in optimization of complex networks. Coming in with a relatively slim knowledge of AIs, I think it became immediately clear that there are some tasks which AIs excel at, but which humans struggle with and vice-versa. Specifically with ZooElite, I feel that the clustering and region based algorithms are very fast and accurate compared to what a human might create. Yet getting the perfect station placement is nearly impossible without thousands of lines of code. This is because there are far too many variables involved in the process, and without implementing computational vision, it is hard for an AI to “see” the options like humans do. Which the data is available, the AI would have to look everywhere around the town, examine what could be terraformed, what could be demolished and where lakes could be filled in to create a station like the one picture above in figure 5.

Where ZooElite AIs excel is where computers have excelled since their inception: quickly performing repetitive tasks. Things like creating vehicles, assigning orders, and building long routes can be done in a fraction of the time that it takes humans to complete the same task. The main reason for this is probably because of the input method. Humans are bottlenecked by the time it takes for them to open a window and visually see the information, then to build whatever it is using their pointing device. I suppose the reason for this is that humans are looking to have fun while playing the game, but still it tends to give AIs the advantage on “boring” tasks.

Lastly, this project begs the question, is there an underlying model for “efficient” transportation networks. After working on this project, I would argue that there is definitely a relationship between creating good transportation networks and graph theory. Using clustering to different granularities and then creating a graph based on spanning trees, then adding back in cycles is a great way to form a general model of a network. Adding in specific data such as city size and geographic features can then help to create a specific plan of action for developing the network. The result will be a very effective
network, which is functional in the real-world. Granted this method is not a replacement for human network planning, but with enough data, models can be created to serve as guidelines for potential routes and optimal vehicle configurations.