Computing Optimal Flow Between Cities

Authors: Trenton Baldrey, Aaron Brooks, Christopher Chappel, Benjamin Mizera

Abstract

Flow problems in computer science have always been of great interest. They can be applied to many different practical kinds of network problems, such as packet flow between routers, and the amount of a substance that can flow through a pipeline. In this project, we investigated a variant of this study of graph theory dealing with the optimal placement of connections in a network to reduce the length of all such connections, and also provide a maximum amount of flow. We attempted to create a program that would simulate the traffic flow between cities in a grid, and then optimize this flow through the modification of the roads connecting these cities.

Introduction

Roads are a very important structure in any civilization. They are the primary medium used to help transport goods and people from place to place, and allow the ability to define how the destinations being connected by the roads interact with each other, and those around them. It is very easy to create a network of roads such that every building is connected to another building, or connected to the entire network for that manner. It is more difficult, however, to design a road network that creates roads to not only effectively link destinations, but does so efficiently. Efficiency here is defined by the trait that the roads are created with the lowest total length possible, using as few resources as possible, while still maximizing the flow of objects, in this case cars, over the roads to allow them to get to their destination.

This problem of efficient and effective road creation and placement is the problem we attempted to address in this project, with arguable success. This problem manifests itself practically as a graph theory problem, where cities are nodes and edges are roads. To measure flow, we introduce the car entity that uses roads to go from city to city. We also introduce the concept of an intersection, which is practically a city that doesn't generate cars, and serves as a middleman for car flow and a destination a road can go to.

Team Members

Overall, we had very hardworking members. Our team was comprised of four individuals: Trenton Baldrey, Aaron Brooks, Christopher Chappel, Ben Mizera. Below we will discuss what each member brought to the project. The complexities of each topic mentioned will be expanded on later in the paper.

Trenton Baldrey was in charge of creating a simple program to create initial city data. He was also in charge of sharing the initial list of cities with all ranks, which included sending this data across multiple ranks. Finally, he was tasked with the creation of the final project writeup.

Aaron Brooks was in charge of running the simulation once all initialization was done. He was tasked with the creation of cars in the graph, as well as the destination of those cars. He also kept track of the performance metrics needed to improve the genetic road creation model.

Christopher Chappel was in charge of the genetic algorithm that we employed to create roads for new generations of the simulation. This task included the ability to create roads between ranks. He was also in charge of aggregating this information to improve the model.

Ben Mizera was in charge of the structure of the simulation. He created the data structures and added the roads to the grid once they were created by the genetic algorithm. He was also in charge of conducting the search over the created graph to provide information to the simulation on where to send cars when running the simulation. Also wrote portions of report pertaining to his implementations.

All team members worked diligently on their pieces of the project. However, there were barriers to effective communication and teamwork on this assignment. The shortcomings of the project and aspects that could have been improved will be covered in a later section of this paper.

Our Implementation

For our approach to this problem, we decided to implement a two dimensional graph without wrap around. This graph would be divided into equal sections determined by the number of MPI ranks the simulation was being run with. We would then read in a file with the positions of the buildings for this simulation and send the pertinent city information to each rank. There, the rank will add the city to their local grid. Each rank is only in charge of "running" their own section of the grid, and similarly to the last homework, there is no global grid, as it may be too large to store in memory.

Once the cities are created, they are not changed throughout the entire simulation. They are a static variable in the simulation, as we are concerned with the placement of roads in the simulation no matter the placement of the cities. Other static variables in this simulation are the number of cars produced, and where those cars end up. We are searching for the road network with the highest amount of cars received by the end of the simulation, with the lowest cost.

Simulation Loop

After the addition of their cities, the ranks enter the primary loops in the program. The outer loop is in charge of the "generation" number of the simulation, while the inner loop is in charge of the "member" number of a specific generation. In each run of the program, we run our simulation a number of times as follows: $r = g * m$. Here, r is the number of times the simulation is run, g is the number of generations in a run of the program, and m is the number of members in each generation. The higher the number of members per generation, the more variable the next generation of roads in the simulation, leading to a higher chance of improving the road network.

Genetic Algorithm

The first step of each rank in each iteration of the inner member loop is to obtain the roads that they need to add to their local graph. To accomplish this, each rank has its own genetic road generator. When a rank needs to generate the roads, the generator uses the positions of the current cities, intersections, and other roads to generate the new set of roads. This is done by attempting to connect one end of a road, starting at one of the above entities, to another entity. If no entity can be found, a new intersection is placed in the graph for the road to go to. In the case that the road desires to connect to an entity in another rank, the generation algorithm first connects the road to a "bridge intersection" that doesn't exist within the grid, but can communicate with it's twin bridge intersection on the other rank. This bridge intersection contains all the details of a normal intersection except a position, as it "exists" in the middle of the road that leads to it.

These roads are then return to the inner member loop of the program to be added to the rank's local graph. Here, the roads are actually connected to the entities they should be, even if they cross ranks.

The Simulation

After the roads have been added to the graphs of the local ranks, the graph needs to have a breadth first search(BFS) run on the grid in all of its entirety. However, since the graph is split between multiple ranks, a simple breadth first search could only be used locally to establish how to get from any city within a rank to another city within that rank. To go beyond the rank required a bit more communication. Essentially, a BFS is run from every incoming bridge intersection, so it knows which cities can get to it within its rank, and how far away each of them are. It then sends this information to it's twin on another rank, where the array containing the ID number of each city and the distance to that city is placed within a map. That data, the cities and their distances, is then communicated to every other incoming bridge intersection, where it is added to their map along with the distance to the cities plus the distance to the relevant outgoing bridge intersection. Unless, that is, the bridge intersection in question can already reach any given city in a lower distance, in which case that lower distance is prefered and it is not added to the map. That updated map is then transferred to an array, sent to the bridge intersection's twin and the process repeats itself. In all, the process of swapping information is carried out a number times equal to the total number of MPI ranks, with each bridge intersection, sending at most 2 integers for every single city within the entire grid. Once that is done, each outgoing bridge intersection communicates to cities within its rank with the cities it can reach along with the distance to those cities. The cities add each of these into a map, along with the ID of the bridge intersection that communicated this, and, of course, replace higher distance solutions with lower distance ones. Once completed, the grid then runs through and adds the addition directions to the intersections within each rank. The ID of the city on another rank is given the same directions as the ID of the bridge intersection that will carry them in the fastest path to the city in that rank.

The reason for the complexity of the searching, mapping and pathing, was the necessity for robustness given some miscommunication within the team. This solution would tolerate any road going between any pair of ranks, sharing that path with all other ranks if it happened to be the one to use.

Another point to this method was limiting the amount of data transferred between ranks. It avoided having the entire grid be shared between ranks, and instead, only relies on the IDs of every city be shared to every road that crosses ranks. So instead of sending every rank an integer for each city (C) and 2 integers for each road (2 *R) (total C*2R), we instead send 2 integers for each city (C & dist) times the number of roads that cross rank(R) . Then we send this to another rank for every single rank (R* C&R * #Ranks). In end, our approach uses more sends, however, it uses less data in each send, so there is some tradeoff in efficiency here. However, given the time we had to write it, we went with what appeared to be the more straightforward approach, favoring ease of the simulation. There was some debate in the group, about difficulty of execution as well, which lead to the given implementation.

This approach also allows minimal effort on the part of the simulation itself, as every "car" can simply know the destination it wishes to reach, and every city will be able to point it to that destination successfully.

After this search is run and the results compiled, we are ready to run the actual simulation. During this simulation, cities generated a number of cars equal to half of their population, which is a number randomly generated when the city data is generated. These cars simply contain their destination's grid ID, and are passed from road to road until they get to their destination. The number of cars that reach their destination during the simulation is recorded and used as a "fitness" factor for use in the genetic road generator.

After the simulation is done running, the results of the simulation are recorded. The two fitness factors discussed in this project were the number of cars received, as well as the amount of resources used to create the roads. As previously mentioned, this data is entered into the next generation of roads, after all members of the generation have been calculated. The final step in the inner loop is to reset the roads, so the next roads can be added correctly.

After the members of the generation are finished executing, their results are sent to each rank's genetic road generator. Here, the fitness results from every member are used to generate the next seed for the new generation of roads to assign to the member loop.

Results, Or Lack Thereof

At this point in the paper it would be expected that there would be some performance graphs showing how this algorithm ran over trong testing. However, our team sadly does not have these sorts of results. For a variety of reasons that will be touched on shortly, our code was not ready to run on the Blue Gene / Q in time to get performance results of any kind. In fact, the code still currently has compile errors.

Analysis of Current Code

Before discussing the factors leading to this unfortunate situation, we will discuss what we perceive works fairly well in the program. Note that the sections that do not work does not speak to the dedication of the designer when implementing the feature. Each member of this team put in a large amount of work to get the project where it is now, and there were too many factors to lay blame on just one part of the project.

Here we will discuss the parts of the program that we have fair reason to believe are working as intended. The generation of initial city information in a document works as intended. The reading of this file and the distribution of this information to the individual appears to work as intended. The generation of roads via the genetic algorithm seems to work in the most common cases, with some possible errors when roads are created that cross multiple ranks.

The breadth first search seems to have the correct logic behind it, as it runs in the serial case of our program. However, it breaks in the multithreaded case, leading us to believe it has something to do with the transition between rank borders. Without the multi rank searching, the multi rank simulations cannot be run and we are therefore unsure if serial or multi rank simulation portion of our program works. Finally, as we were running out of time to complete the code, we did not implement the second portion of the genetic algorithm. We therefore did not have the ability to combine the fitness data from the member runs to create a new seed for the creation of subsequent roads. Also worth noting the sim was never tested in the single or multi rank execution, giving no determinate of its working capability.

Retrospective Improvements

Now that we have looked at the current state of the project code, we would like to spend a little time discussing some of the factors leading up to this less than desireable outcome. We as a team take full responsibility for not accomplishing this project and this section simply serves as a way we can reflect on the circumstances and events that have led us to this point.

Vaguely Defined Project Scope

Even though our team met with Professor Carothers on multiple occasions to go over the scope our our project, it appears that we still attempted to "bite off more than we could chew." We wanted to create the entire code base from the ground up, and envisioned a system in which we would be able to run the simulation in multiple different ways, such as with a combination of one way and two way roads.

Looking back, it would have been wiser for us to have defined a narrower beginning scope for this assignment. As it is, we have written nearly 2,800 lines of code between the four of us, and spent many hours coding and debugging. This time could have been better utilized at the beginning of the project defining exactly what we desired to get done in the project.

Poor Team Communication

This is a very personal flaw for the team. It was often the case that team members would work on and fix their code, test their code, and then the next party whose code needed addressing would not hear of this desire for a while afterwards. There were many of these small communication errors between all members that lead to a large decrease in productivity. There were no clear checkpoints defined by the team at the start of the project to guide members and help define a timeline for completion.

It is also worth mentioning that all teams had one fewer weeks to prepare this final project than what was defined in the syllabus. This would probably not have mattered for our team, as we had an extension to this assignment and still do not have running code. However, it would have been nice to have the extra time, just in case.

Lack of Previous Research

As mentioned previously, our team desired to write all code for this project from scratch. This may not have been the best idea, as there is surely code relating to the flow of of graphs available even as a template to base our code off of. Doing more extensive research into the problem we were trying to solve would have allowed us a better understanding of how to approach it.

Predictions of Execution Time

Even if we do not have current results for our code, this does not inhibit us from reasoning about the performance of our code. While we do not have the expertise to provide concrete estimates on execution time, we can reason about the time spent in each section of our code.

Initialization

To start us off, we will note that we approach the problem of distributing initial city information to other ranks from a more "master and slave" perspective. Rank 0 is in charge of reading all the information from the initial city file and sending out this information to the appropriate ranks for them to create their cities. This may incur a loss in initial performance. However, the creation of the cities in the graph is trivial after the data has been distributed.

The next computationally expensive piece of code is the genetic creation of the roads in the member loop of the program. The loop that creates the roads in this generating function needs to have three flags set to true to be able to exit: there must be a connection to the above rank, below rank, and all cities within the rank must be connected. This ensures that the entire graph is connected in some way. However, there is a small chance that this algorithm takes a very long time to finish, as it randomly assigns roads to cities and intersections. The primary cost in this portion of the program would be in execution time, though there is some IO being done between ranks as roads are created that cross rank boundaries.

Simulation

In the BFS, normal BFS algorithms is O(V + E) where V is the number of vertices and E is the number of edges in the graph. However, the algorithm used in this program does not guarantee this sort of runtime, and while it may not run as long as the genetic algorithm, it will still make an impact in execution time. The IO being done here is also of note, as the BFS must traverse all ranks to obtain all the information needed. This is also a relatively execution time intensive procedure.

However, arguably the most computationally intensive portion of this code would be the running of the actual simulation. In the simulation, we have to update each car currently in the simulation, send cars across ranks if needed, and generate new cars. After each tick, a barrier is called to ensure that all ranks are running as close to in parallel as possible. This section would more than likely involve the most point-to-point IO usage of any of the sections, as cars are moving across ranks quite often and the receiving rank must be notified.

Strong Scaling Prediction

In general, while this code would more than likely benefit from increased ranks and cores, it would more than likely not scale as well as desired. The amount of extra work that each rank must do in terms of road generation and simulation as as each new rank is added would most likely decrease the gains from increased processor and rank count.

Related Work

Despite our lack or results in this project, there have been other studies done around parallel computing and traffic flow, many of which have found some very positive results and improvements in algorithms because of their simulations. This makes sense, as traffic flow is such an important problem.

For example, there was a study done involving microscopic traffic modeling using parallel computing. In this study, the authors viewed traffic flow as a one dimensional problem, and therefore employed a single bit encoding scheme. This allowed the authors to reach utilization levels of near 100%, as well as speed up the simulation by a factor of 1000x! (Nagel, K., and A. Schleicher.)

In a study much similar to the one we as a team were trying to undertake, (Nagel, Kai, and Marcus Rickert) the authors of the paper parallelized TRansportation ANalysis and SIMulation System (TRANSIMS) traffic micro simulation. They implemented this parallelisation by assigning CPUs to specific geographical regions, much like we were segmenting the grid in our simulation between ranks. The TRANSIMS system implemented many of the features that we had barely thought about, such as signaled and unprotected turns, speed of cars driving down roads, and even lane changing.

The authors of this paper decompose the sections of roads into similarly sized pieces for each CPU to govern. The CPUs can then gain the needed information from their neighbors and increment the events in their own section. In the end, the authors found that their implementation was able to speed up a specific network (the Portland 20024 links network) ran 40 times fast than real time when 16 500 MHz computers were connected with 100 Mbit ethernet.

Summary and Future Work

In summary, our team attempted to address the problem of traffic flow between cities by genetically generating roads between cities. This program, despite extreme efforts, did not prove feasible to code completely in the time that was allotted. We have explored what the purpose of the code was, and how it would ideally work. We have also discussed the possible factors behind this failure, as well as what can be done in the future to prevent this type of situation from presenting itself again.

Finally, we discussed the runtime of the code if it were to be working, making note of execution intensive and IO intensive operations. We then examined some works done by other authors that were similar to the project undertaken here.

In terms of future work that could be done in this field, and with our program, there are many options. Adding the ability for cars to have an inherent speed that allows them to travel to their destinations in different times would make the simulation more realistic. Another feature that could add a sense of realism the simulation is the addition of “rush hours” or times in the day when more traffic is present on the roads.

When everything was said and done, this project was an enjoyable challenge. Working in this class this semester has given us all a greater respect for the intricacies of parallel programming, and has given us much needed experience with parallel computing.

References

Nagel, K., and A. Schleicher. "Microscopic Traffic Modeling on Parallel High Performance Computers." Parallel Computing 20.1 (1994): 125-46. Web.

Nagel, Kai, and Marcus Rickert. "Parallel Implementation of the TRANSIMS Micro-simulation." Parallel Computing 27.12 (2001): 1611-639. Web.