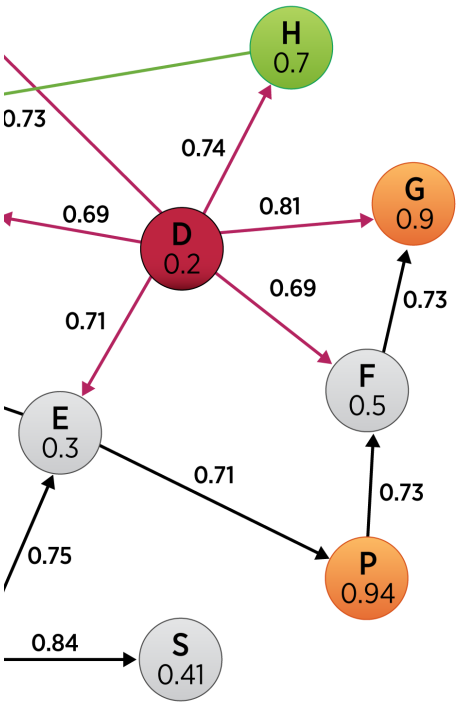
DEFEATING BRAESS' PARADOX

BY USING DIJKSTRA ALGORITHM

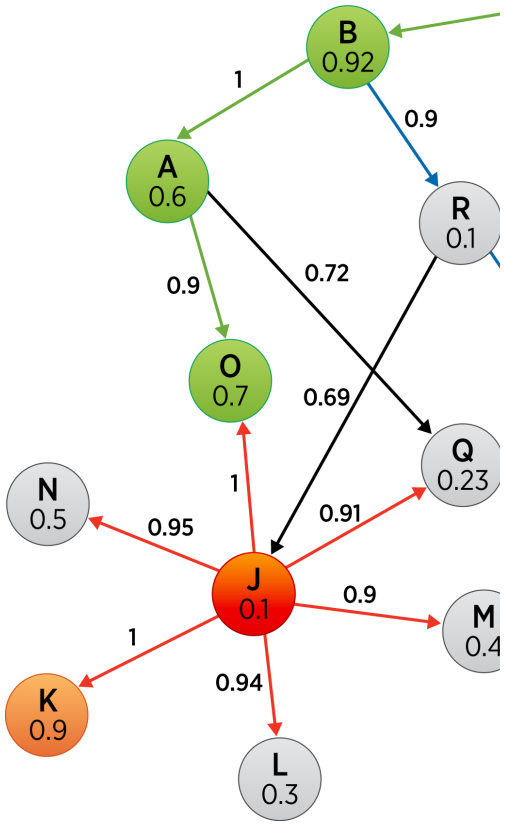
TO EVALUATE ROAD INFRASTRUCTURE CHANGES



Hayat AlHassan

Peter Chen

Theodore Tenev



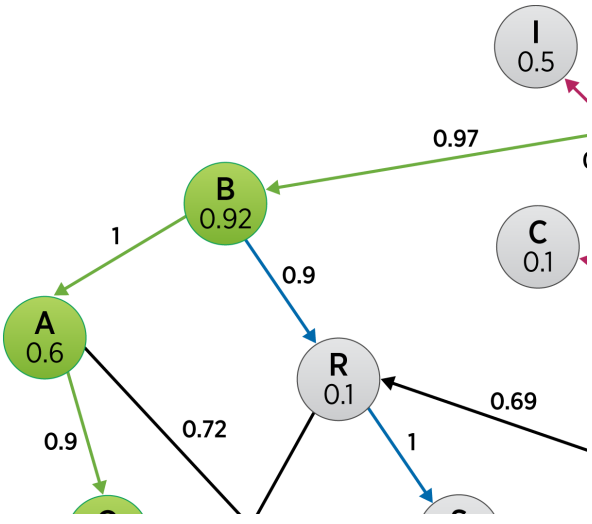
Professor Yaw Nyarko

ECON-AD 217

Technology And Economic Development

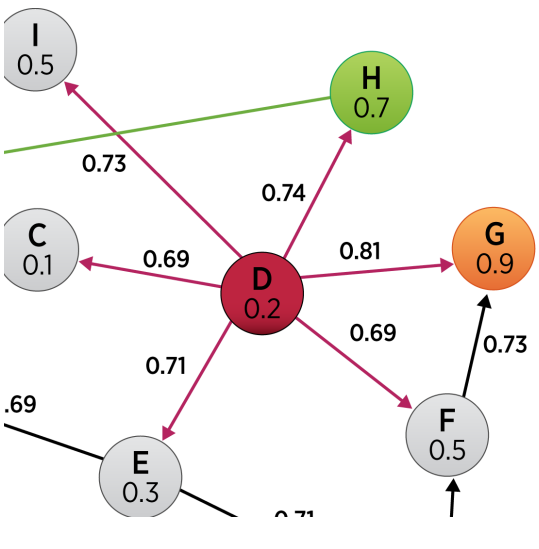
May 8, 2017

Table of Contents:



I. INTRODUCTION

Our team has identified an area of our daily lives which is often overlooked and by most people as it is regarded as a commodity and a status quo that cannot be changed. Namely, we looked at the relationship between road traffic and road infrastructure. More often than not, road infrastructure changes are made without consideration of a simple, yet counterintuitive principle – Braess' Paradox. Hence, our team decided to develop a simple interactive road traffic simulation, through which the user will be introduced to the concept of the paradox. We want to empower the user to see how a specific decision about road infrastructure change affects the status quo by presenting the information in a visually rich yet simple manner. The ultimate purpose of our Traffic Simulation Game is to enable governments and third-parties to make informed decisions about making changes to existing road infrastructure. Ultimately we strive to create a consistent way to test proposed changes and evaluate them according to social benefit and economic prudence.



II. MOTIVATION

We have been demonstrated in class that often times an economic decision that strives to improve the existing conditions and introduce social benefits, while seeming sound and well-reasoned, can in fact have a negative effect on society and make things worse off. In the particular case of building a new road with the intention to alleviate traffic but in fact increasing overall journey time, German mathematician Dietrich Braess identified the paradox for the first time in 1968. Since, it has been described as Braess' Paradox.

In terms of Game Theory, Braess' Paradox demonstrates a tangible case where the Nash Equilibrium may not result in with the best overall usage of a network, nor an optimal outcome for all players given the previous conditions. A change in the network of roads produces a new game structure, which leads to a situation similar to prisoner's dilemma. In the original Nash equilibrium, drivers had no incentive to change their routes. When a new road is built, under Braess' paradox, drivers will continue to switch their roads until they reach Nash equilibrium despite the increase in overall journey time.

One particular example in Braess' Paradox in real life is the closure of 42nd street in New York City in 1990. While some might expect the closure of the road to worsen traffic conditions in the neighborhood, in fact it reduced the amount of congestion in the area.

III. PROJECT DETAILS

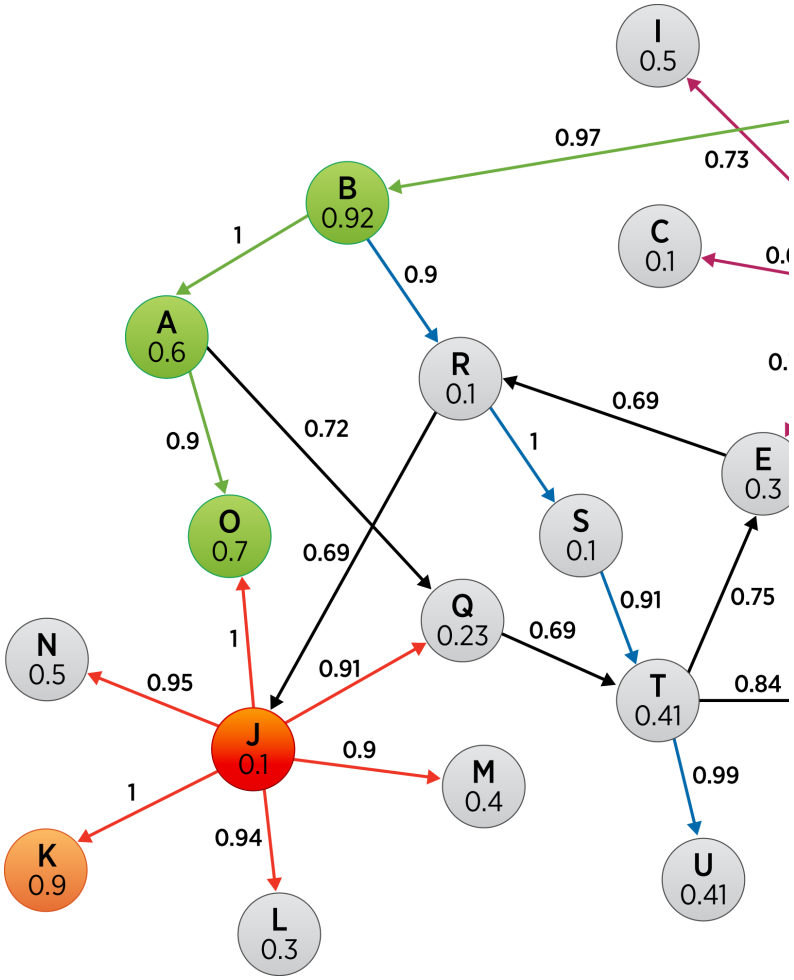
Approach

Our project undertakes an approach similar to hypothesis testing in statistics. The simulation provides the user with a visual representation of the road network. Given the amount of traffic/load of each road on the network, upon constructing a new edge our simulation will be able to confirm or deny the hypothesis whether this new road will reduce overall journey time. The simulation will first calculate the shortest path given a beginning point and an ending point for the network and display the total travel time and the path itself. Then, upon a change in the road network (via the creation of a new connected node, or a new edge between existing nodes), our simulation will calculate the new shortest path given a beginning point and an ending point for the new network and display the new total travel time. Ultimately, the user will be able to compare the first shortest path's travel time and the new shortest path's travel time, in order to decide whether the newly-created road reduces the total travel time or not.

Core Application Features

- Ability to create a custom graph and manually input the traffic for each edge, in order to facilitate representation of various dynamic environments.
- Calculate initial total journey time, given a number of travelers
- Ability to add/remove roads on existing graphs
- Identify new Nash Equilibrium after addition/removal of a road
- Calculate new traffic loads for each edge
- Calculate new total journey time

- Display an explicit advice on whether the addition/removal of a road is socially-beneficial



IV. FRONT-END DESIGN

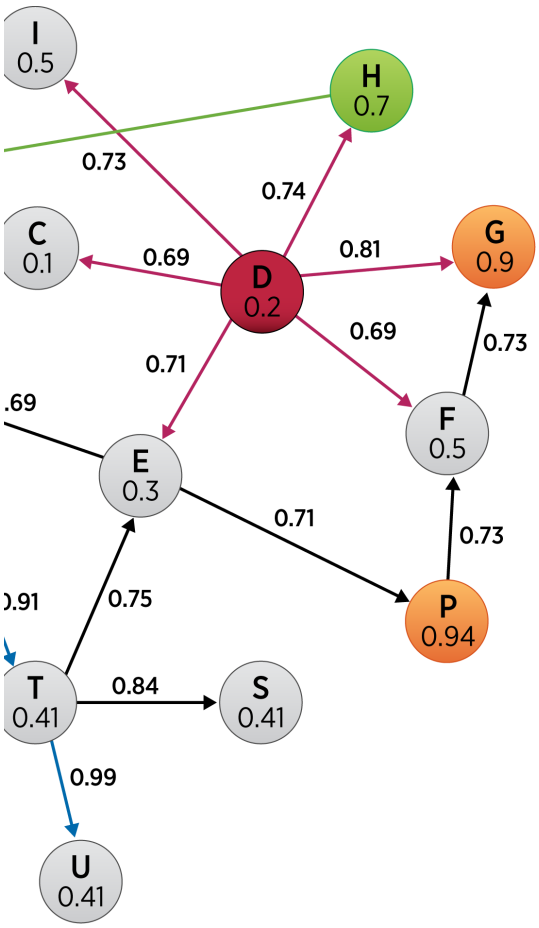
JavaScript Libraries

Our Traffic Simulation Game aims to provide an experience that would allow users to witness the effects of their decisions in real time. Through this simulation we aim to provide a better understanding of network and game theory to the player through an interactive interface, specifically within the context of road networks. While Peter Chen will be working on developing the backend of the game, I will be focused on the design of the interface and the user experience. There are two ways that could be used in developing the frontend application. On the one hand a JavaScript library called Cytoscape.js could be employed. This library is an open-source graph theory that enables the implementation of graph analysis and visualization. The use of this tool would ease the process of visualizing the graphs, but it would also add some level of constraint. On the other hand, the frontend simulation could also be built on top of more general JavaScript graphic libraries, such as P5.js. This would allow for more flexibility in the visual simulation of traffic, but would also require a higher level of complexity.

User Experience

When users first get to the website they will have three options, to start the game, walk through a tutorial to understand how the game works, or to read about the game and the theory behind it. At this point, you might be wondering how the game goes, the following explains how it is played. When you choose to start the game, you will be asked about the level of difficulty you would like to take on. You will then be given a graph consisting of nodes and links, representing roads. An identified number of cars will be expected to

move from the starting to the ending node. The links will be marked with values and variables identifying the speed at which the the cars can move. With the knowledge of Game Theory the user will have the choice to build new roads, more specifically connect nodes with new links. Depending on the user's answer a feedback will be returned with a short explanation of why the answer is correct or wrong.



V. BACK-END DESIGN

Considerations

In our application, we aim to allow users to interact with preset models. These will be presented in the form of levels in a game, where users try to get from one node to another by choosing the best ways to build a route via other nodes. Each edge between two nodes will contain a weight, which is equal to the cost of travelling on that edge. In order to achieve the above, I have two options:

1. To hard-code different levels into the application
2. To create an algorithm that will generate suitable scenarios

The first option will be easier in terms of the predictability of our results, and is easy to control and monitor. However, the code in this option will not be easily malleable, posing future challenges for anyone who wishes to extend the levels of this game, as they will have to manually create new levels.

The second option, on the other hand, requires more preparation before the actual development. I would have to control for different parameters that would affect the outcome of the network traffic scenario. These parameters may include the number of cars, the number of nodes, the number of edges, which nodes are connected, and the relative weights of those edges. If an appropriate algorithm can be designed, however, this will allow for an infinitely more powerful game as the levels can go on and on. Therefore, I am currently researching the possibilities of devising such an algorithm to be implemented in our application.

Integration with Front-End Library

Some further considerations to be taken into account are the frontend tools that are employed. We have discovered a JavaScript library called cytoscape.js that is designed for network graphs. Cytoscape provides a plethora of functions in their API so that it will be easier to implemented network graphs on a web application. While this definitely makes our lives easier as developers, it also introduced certain constraints, especially to the backend. More specifically, while the logic may not change, the implementation will be modified along with our choice of frontend library.

Game Logic

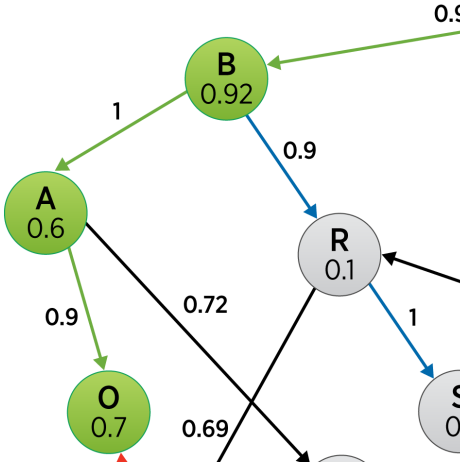
High level description

In the background of the simulation, we will use *Dijkstra Algorithm* to find the shortest path from a given starting point (aka. source) to all destinations in a directed graph (single source shortest path problem). During this process we will also determine a spanning tree for the graph. We can use this then to find the to find the shortest path from the source to *a specified* destination in the graph. Upon a modification to the graph (by either creating a new connected node or an edge between existing nodes by the user), we will run *Dijkstra Algorithm* again to find the new shortest path from the source to the same destination as before. Meanwhile, the program will calculate the total weight (representing the travel time) for the initial shortest path and the new shortest path and display both.

Further Algorithm Details

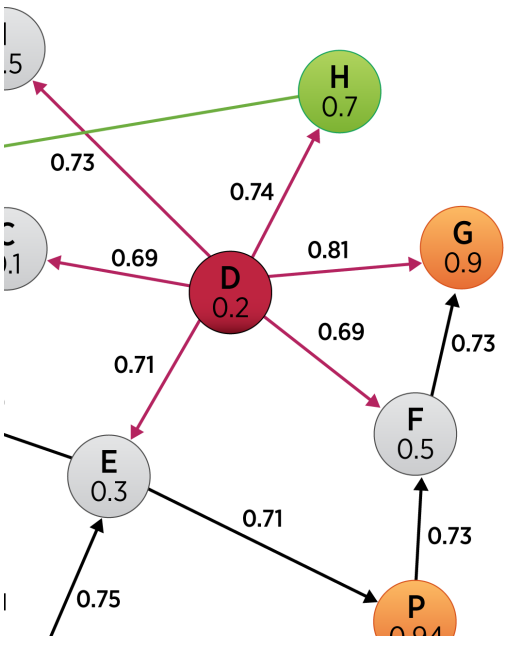
The idea of Dijkstra is simple. Dijkstra partitions all nodes into two distinct sets: unsettled and settled. Initially all nodes are in the unsettled sets, e.g. they must be still evaluated. A node is moved to the settled set if a shortest path from the source to this node has been found. Initially the distance of each node to the source is set to a very high value.

First only the source is in the set of unsettled n nodes. The algorithms runs until the unsettled nodes are empty. In each iteration it selects the node with the lowest distance from the source out of the unsettled nodes. It reads all edges which are outgoing from the source and evaluates for each destination node, in the edges which are not yet settled, if the known distance from the source to this node can be reduced while using the selected edge. If this can be done then the distance is updated and the node is added to the nodes which need evaluation.



VI. CONCLUSION

We hope that our application will be of some real-world use. Through this application we hope to educate people about game theory and the implications that it has on the real world. We hope to provide them an experience to understand first hand why an investment in infrastructure is not always a good investment. We hope that through the development of the game logic, the core components of the application will be designed in such way that it could be used as a foundation for a program that allows for customizing cases and scenarios and simulating how effective building a new road will be. By using the principles of Game Theory, we empower the user to avoid misinformed decisions and not fall into Braess' Paradox.



V. REFERENCES

Vogel, Lars. "Dijkstra's Shortest Path Algorithm in Java." *Vogella.com*. Lars Vogel (c) 2009, 2016 Vogella GmbH, 29 Sept. 2016. Web. 06 May 2017.

Easley, David, and Jon Kleinberg. "Chapter 8: Modeling Network Traffic Using Game Theory." *Networks, Crowds, and Markets: Reasoning About a Highly Connected World*. Cambridge: Cambridge UP, 2016. 229-47. Print.

The full code of our application can be found on our github page:

https://github.com/vcheeze/Network-Traffic-Game