Braess' Paradox and Imperfect Information

Bagatur Askaryan, Laurence Calancea, Theo Walker

December 10, 2018

Abstract

Ostensibly, a new route that is introduced to a network, with low associated cost, should improve the performance of agents in the network. This intuition is upended by Braess' Paradox, the phenomenon that occurs when the addition of such an edge results in a user-optimal equilibrium that is Pareto-dominated by the equilibrium of the original graph. In this paper, we explore the paradox in detail using a canonical graph, and construct simulations that provide realistic models of the paradox occurring. We focus on studying how the magnitude and presence of the paradox is affected by the information available to agents and by the transition from the equilibrium flow to the end state of a sequential move game. We find that the paradox is exacerbated (i.e., becomes more costly) in settings of less information, and that a predictive agent can actually achieve the socially optimal flow faster (at lower θ values) than the equilibrium flow.

1 The Paradox

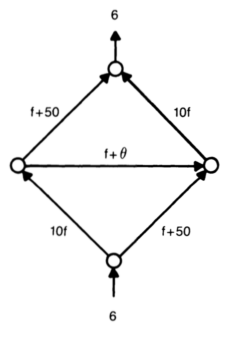


Figure 1

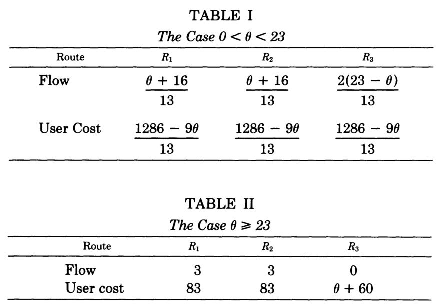


TABLE I

The Case $0 < \theta < 23$

Route	R_1	R_2	R_3
Flow	$\frac{\theta + 16}{13}$	$\frac{\theta + 16}{13}$	$\frac{2(23 - \theta)}{13}$
User Cost	$\frac{1286 - 9\theta}{13}$	$\frac{1286 - 9\theta}{13}$	$\frac{1286 - 9\theta}{13}$

TABLE II

The Case $\theta \geqslant 23$

Route	R_1	R_2	R_3
Flow	3	3	0
User cost	83	83	$\theta + 60$

Figure 2

The network in Figure 1, sourced from Steinberg and Stone (1988), describes the general form of a graph which can experience Braess' paradox. The culprit is the center edge with associated cost $f + \theta$, where f is the flow across the edge (i.e., the number of agents who decide to use it) and θ is a parameter that can be varied to achieve different equilibrium flows. As the table in Figure 2 indicates, when modeling the behaviour of 6 agents, the paradox occurs for $\theta < 23$; increasing the cost of the central edge up to this point *decreases* user costs as a whole.

2 Agent/Information Models

The Greedy Model

Our first model is greedy, in the sense that agents are modeled as choosing their paths based on which neighboring edges are cheapest. This emulates the setting in which agents have no information about the traffic network except what is immediately apparent to them. A greedy agent begins at the start vertex and at each step, chooses her next edge based on the edge with the cost that is currently the cheapest. Greedy agents know how many agents have already made their decision on the network, and they know the present flows for each edge they come upon, but they do not know the layout of the entire network, how many agents will come after them, or what paths these agents will choose. This kind of assumption is fairly realistic for settings with a high proportion of drivers that are unfamiliar with the setting; such drivers will often take whichever road seems to be the least occupied at a given moment, disregarding the fact that this might not correspond to the shortest path.

The UCS/A* Model

The UCS/A* model adds complexity in the sense that instead of greedily choosing the cheapest edges in the graph, agents choose the cheapest path. This is done through an implementation of Dijkstra's Algorithm, which maintains a minheap where vertices are added when they are found to be reachable from a previously explored vertex and prioritized based on the weight of the distance from the starting vertex to that particular vertex. Agents in this model have perfect information about the graph structure (i.e. the cost functions associated with each edge) and perfect information about the flow through the network at the time of their decision, but no information regarding what will come after them. This is an approximate way of modeling drivers who are familiar with the traffic networks they are traversing and are updated about traffic up to the period in which they are about to drive, but have no way of knowing how many other people will begin driving in the same and subsequent periods. They have a snapshot of traffic right as they head out, but no knowledge about future traffic dynamics.

The GMaps (Predictive) Model

Another class of results we obtained were for the actual user-optimal equilibrium flow of the network. While the Greedy and UCS/A* approaches are certainly informative and are likely to model real-world flows effectively, determining the actual user-optimal flow in the network is more complete in the sense that no agent would benefit from having changed her decision. This is relevant when the applications we are considering involve complicated networks with traffic flow in many different directions, so that decisions can in some sense be "undone" traversing the graph in the reverse direction. Given a graph and the relevant parameters, including θ, we compute the user-optimal flow through an implementation of the Simplex algorithm.

We then introduced uncertainty into the system by combining the ability of the LP formulation to solve for the actual equilibrium with our sequential-move game model (which we previously used for the Greedy and UCS/A* agents). The LP formulation uses the total number of people traversing the graph as a parameter when obtaining which paths will be in the equilibrium and which paths will not. In this sense we decided to add randomness by allowing the users to predict what the total number of people on the graph will be. In turn this allows them to make an informed decision about the paths that will likely be in the equilibrium. For simplicity we chose a binomial distribution to reflect the user's prior distribution. We assumed a binomial because it has a shape similar to a normal except that it is discrete. To summarize, this last model is of agents who know the network structure, the past flows, and a predicted distribution for total flow. These agents then use previous flow to find the conditional distribution for final total flow, and using this find the expected final cost of each path they could take.

Agents calculate the best path to take, and its associated cost, for each possible value for the number of total agents n. These costs are then weighted by the probability of the relevant value of n occurring. In this way, the expected cost of a decision is minimized and utility is maximized. If multiple paths have the same expected cost, the chosen path is the one with the lowest current cost. If there are multiple such paths one is chosen at random.

This method is used to model a service like Google Maps, which has up-to-date traffic information and (perhaps using historic data) some distribution for how much more traffic there is to come. It uses this information to come up with an expected best path.

3 Implementation

High-level Overview

The simulation takes agents and has them each choose their path sequentially. Once an agent is committed to a path they cannot change it, and all agents are ultimately sent in together. The simulation returns the final state of the network, including the average cost per agent.

The simulation takes the following parameters: network (graph), θ, number of agents (i.e., total flow), type of agent (i.e., search type), and resolution. The network has up to one edge whose cost is dependent on θ and only linear (in flow) edge costs. For most of our study we used the one canonical graph, but the simulation can be used for any directed graph and particular (origin, destination) pair.

The resolution is used as a way of approximating continuous flows, which is how the equilibrium flow is calculated. Higher resolution equals smaller unit of traffic flow being sent at a time, and therefore close to continuous results.

The type of agent determines how each agent will search for an optimal path when it is their turn to choose. The three options are Greedy, UCS, and GMaps, as explained in section 2. At present only agents of the same type can be run against one another, but this can be relatively easily changed (which would make for an interesting future study).

The Greedy and UCS search methods are fairly straightforward - they calculate exact costs on the edges at the time of the agent's decision (that is, using the flows of all preceding agents), and either greedily or optimally find the shortest path, respectively. The GMaps method is more complex. To find an optimal path, the kth GMaps agent has to first take the general distribution on potential total flows and find the conditional distribution, given they are the kth agent to go. For each potential total flow value (above a certain minimum probability threshold), this agent must then calculate the equilibrium flow on this particular network. This requires an LP solver and an automated LP formulator. The optimization objective is to minimize total cost, and the constraints are flow conservation for every vertex, all paths that are used having equal cost in equilibrium (Wardrop's Principle), and flows on an edge being the sum of the flows along each route which contain that edge.

Codebase

A quick overview of the codebase:

The **plot.py** file creates plots of the cost for particular search algorithms as well as multiple value of total number of users as a function of the parameter θ.

distribution.py file obtains the conditional distribution for the number of total people on the graph given the number of previous users. It does so by sampling many values from the original distribution (Binomial), then creating a frequency dictionary for each variable and then obtaining an estimate of the conditional probability as the fraction of the frequency of a number over the total number of samples. In the end the file computes the expected cost for all the possible paths given a particular number of previous users of the graph.

The **findpaths.py** file obtains the total possible paths from a starting point to an end point by iteratively performing multiple DFS.

The **heap.py** file creates a modified version of the heap found in the heapq package. We use the heap when performing the search for the lowest cost path in the Uniform Cost Search.

The **equilibrium.py** file obtains the paths that can achieve an equilibrium using the LP solver and their cost, as well as calculates the cost for the paths that cannot be part of an equilibrium in the LP (this is because the LP assumes that all paths in the equilibrium will have the same total cost, which is not possible for some paths). The cost for the paths that cannot be part of the equilibrium is obtained by summing the cost of each edge in the respective path after it's cost was updated by the equilibrium flow.

The **search.py** file implements the Uniform Cost Search, the Greedy Search and the Google Maps Search (which uses predictions about the number of total agents).

cmd-line.py allows us to run custom simulations from the command line.

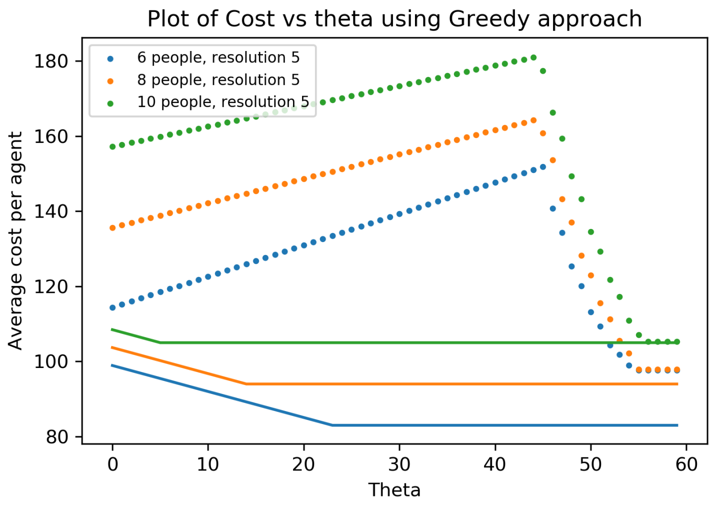
The **lp.py** file is a Linear Program solver that was provided in CS 124 and whose functionality we used in our project.

The **simulation.py** file performs the desired search for a given resolution value, number of users, θ value.

4 Results

Greedy

Figure 3

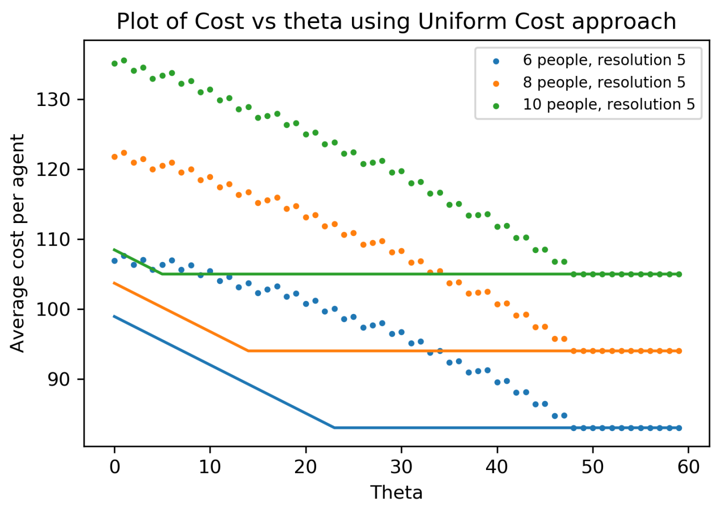


To test how letting greedy agents sequentially choose their paths affects the Braess' Paradox, we ran the simulation with 6, 8, and 10 agents on the canonical graph, for $\theta \in [0, 60]$. In Figure 3 you can see the results: the dotted lines are the average costs per agent in the simulation, and the solid lines are the equilibrium costs.

We can very clearly see in this graph that not only does Braess' Paradox still occur in the Greedy setting, but it is actually augmented, both in magnitude and persistence. Whereas in equilibrium the socially optimal flow is reached between $5 \leq \theta \leq 23$ for the different total flow values, in the Greedy setting it is not until $\theta \approx 55$ that any of the average costs bottom out, and even then it is not always at the socially optimal flow that they do bottom out. Moreover, the average costs in the Greedy settings before achieving the socially optimal flow are always greater than the equilibrium costs, except for the 10 person setting where they are strictly greater.

It is also interesting that the average costs in the Greedy setting are increasing up to the point when they finally drop all the way down. However, this is simply a feature of this particular graph, and is not necessarily indicative of a broader result. In general, we should be careful not to extrapolate too much from the Greedy results because they are very graph-dependent. That being said, it is still very clear that the Greedy setting can maintain and indeed significantly augment Braess' Paradox.

Figure 4



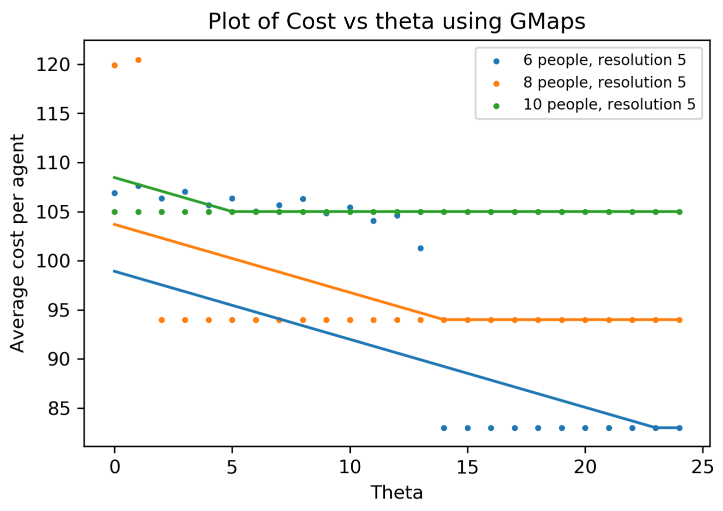
UCS/A*

In Figure 4 we see the results of the same set of simulations as in the Greedy setting, only now using UCS agents instead. Again the dotted lines are the results of the simulation and the solid lines are the equilibrium flow calculated using the LP. We can see that in the UCS setting the results follow the equilibrium costs much more closely than in the greedy setting, but that they still augment the effect of Braess' Paradox. We can see that in all three settings the socially optimal flow is reached (unlike in Greedy), but that it is only achieved at $\theta \approx 48$. Up until they achieve socially optimal flow, the avg costs in the simulated settings are strictly greater than the equilibrium costs, and the difference between average cost and equilibrium cost is positively correlated with the number of people. The simulated settings also display an interesting concavity and a startling similarity, something which warrants further study.

GMaps

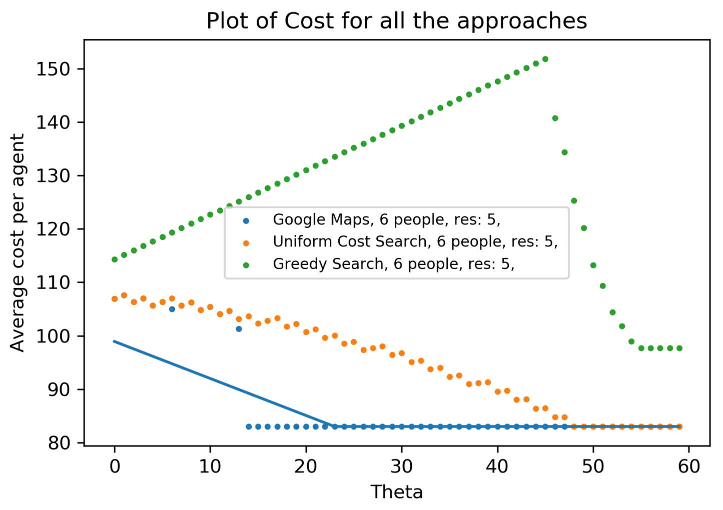
For the Gmaps Search we notice that the equilibrium costs are achieved for lower starting value of θ. Interestingly, in this search we also have that the average cost obtained is sometimes lower than the equilibrium cost. This can be explained by the fact that with the additional information of agents approximately knowing how many people will be using the graph, as well as which paths would be found in the equilibrium solution, they can make more informed choices and not take paths that deceivingly seem appealing in the beginning. This is a socially optimal solution that reduces the average cost.

Figure 5



Comparison

Figure 6



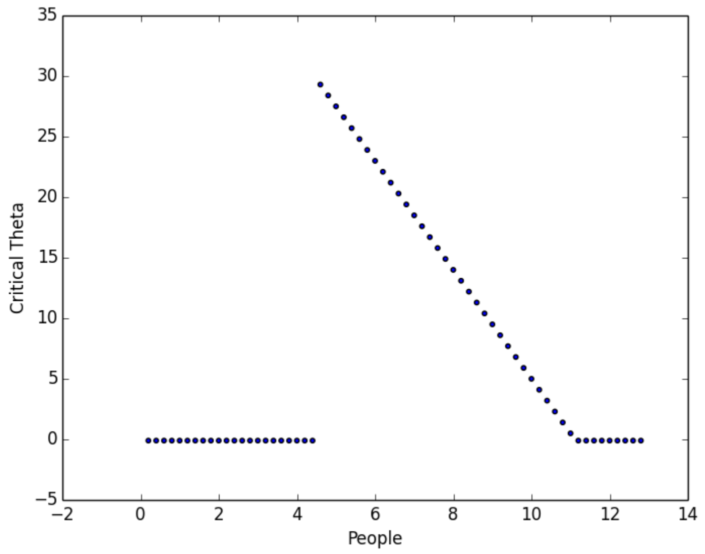
In the last suite of simulations we compare each of the different types of agents with total flow 6 (the equilibrium flow in this setting is the solid blue line). We get several striking results from this comparison: first and foremost, we see that agents with more information always do weakly better than agents with less information. That is, GMaps beats UCS beats Greedy for every single θ value. We also see that agents with more information achieve the socially optimal outcome faster, up to the point that GMaps even beats the equilibrium flow to the socially optimal outcome (it helps establish and maintain collaboration among agents in this way). What would be interesting to further study is how the costs seem to become more and more convex as information increases.

5 Conclusion and Further Research

Our results indicate that information plays a crucial role in the presence and magnitude of Braess' Paradox. We can see that in settings of imperfect information the paradox does still occur and is in fact augmented, in that it occurs for larger θ values and the avg costs are greater than the equilibrium costs. We also see that if we build a predictive model we can actually begin to counter the paradox and achieve the socially optimal outcome before the equilibrium outcome does so.

While we were able to cover a lot of ground in our research, there are many other directions that this work could improved and expanded. Most immediately, we realized that our implementation of the GMaps Model causes agents to make many redundant calculations. Each agent individually computes the equilibrium flows for every possible value of the number n of total agents. This means that our simulations would see an increase in efficiency if we dynamically stored these results and allowed them to be accessed be all agents.

Figure 7



One area that we are interested in exploring is empirical evidence in our simulations for Anna Nagurney's proof that Braess' paradox disappears under high demand.

We were able to scratch the surface of this topic through the use of our LP equilibrium flow solver, by writing a program that iterates over different numbers of agents and, for each number, computes the value of θ for which the paradox vanishes. The results are plotted in Figure 7. (Note that critical θ values of 0 correspond to both $\theta = 0$ causing the paradox to disappear, *and* it being literally impossible for the paradox to manifest itself in the current graph structure.) As expected, the higher the number of agents, the lower the value of θ needed to be for the paradox to disappear, and ultimately for $n \geq 11$, it was impossible. The relationship in the figure is linear, and one way we could investigate this are further is to determine whether this linear relationship holds more generally, either when we introduce uncertainty as in the Gmaps Model or for more arbitrary graphs (note that our LP solver can take as input an arbitrary graph).

The binomial distribution used in our GMaps simulations is well-behaved in the sense that it takes a similar shape to a normal distribution. It is also realistic because it models a fixed total number of agents and each of these agents as independently deciding whether or not to participate in the network (i.e., it consists of a series of independent Bernoulli trials). However, there are reasons that are just as valid for modeling the total number of agents as a random variable following some other distribution. Our code can easily adjust for that by sampling from the desired distribution instead. We would expect different distributions to give different, unique results for the simulation.

Another direction we could go in is running different types of agents against each other, as mentioned in the High-level Overview section. This is analogous to the trials in Problem Set 2 in which we paired different kinds of BitTorrent clients against each other and measured individual and overall performance. As mentioned earlier, our simulation code has been written flexibly so that implementing this would be doable.