Autonomous Vehicle Intersection Management

Final Paper for CMSC499A Independent Research

Robert Adkins

Spring 2016

Faculty Advisor: David Mount

Abstract

This paper explores the use of game theory to design an efficient algorithm for routing self-driving cars through intersections. Despite the discouraging exponential naive solution to this problem, there are some efficient optimizations which are exposed.

Contents

1 Introduction

Tesla's [1] and Google's [2] recent innovations in self-driving car technology have helped pull the widespread use of autonomous vehicles into the not so distant future. With the technology advancing so quickly, it is natural to consider how these vehicles will make decisions about their trajectories through various road networks. In particular, intersections are an interesting place to begin these considerations. Will cars act purely independently to navigate through intersections accident-free? It seems unlikely. It would be more effective to allow vehicles to communicate with one another in such a way that promotes a balance between social and individual benefits. This idea leads to the motivation of using the tools of game theory for designing an efficient algorithm.

2 Background and Definitions

It will be useful to go over some basic concepts of game theory before continuing. The content of this section was studied mainly from [3] and [2].

Oftentimes situations arise in which individual agents intend to compete against each other to achieve personal objectives. *Game theory* is a mathematical field which models these kinds of competitive situations and provides methods of calculating the asymptotic tendencies of such systems as time progresses. A game can be formally described as follows.

Definition 2.1. A *game* $G = (n, S, U)$ is defined by n players labeled 1 through n, a set $S = S_1 \times S_2 \times \ldots \times S_n$ of strategy profiles where each S_i is the set of strategies available to player i, and a set $U = \{u_1, u_2, \ldots, u_n\}$ where each u_i is a map from an n-tuple $\hat{s} \in S$ of strategy choices for all players to a real-valued utility for player i.

With this model, the players are placed in a context wherein they choose strategies from their strategy sets and subsequently measure their personal gains through evaluating their utility functions on the strategy choice vector containing all player decisions.

Within a game, it is necessary to define some sort of situation in which the players are happy with their current choice of strategy. We wish to define a state of balance in a given game that says when this occurs.

Definition 2.2. We are given a game $G = (n, S, U)$ as defined in Definition 2.1. An *equilibrium* is some kind of strategy recommendation to players that is *self-enforcing*; i.e., no player i can improve the value of its utility by deviating from his recommendation.

This self-enforcing constraint takes on different interpretations depending on the equilibria in question. Pertinent to this paper is the correlated equilibria. Though for completeness, we start with the pure Nash equilibria.

Definition 2.3. A *pure Nash equilibrium* is an equilibrium that occurs when players are recommended to deterministically play a single strategy all of the time.

[1] http://www.wired.com/2015/10/tesla-self-driving-over-air-update-live/
[2] https://www.google.com/selfdrivingcar/

Given a strategy profile $\hat{s} \in S$, denote $\hat{s} = (\hat{s}_{-i}, s_i)$ where $\hat{s}_{-i} \in S_{-i} = S_1 \times \ldots S_{i-1} \times S_{i+1} \times \ldots \times S_n$, the profile which omits the ith player's strategy choice, and $s_i \in S_i$. The purpose of this notation will become clear in subsequent definitions. The following is the formal representation of the self-enforcing principle in a pure Nash Equilibria. Given a pure Nash equilibria $\hat{s} = (\hat{s}_{-i}, s_i) \in S$,

$$\forall i \in [1, n], \forall s_i' \in S_i : \quad u_i(\hat{s}_{-i}, s_i) \geq u_i(\hat{s}_{-i}, s_i'). \tag{2.1}$$

In other words, no player can improve his utility by unilaterally switching strategies from the recommendation in $\hat{s}$. Pure Nash equilibria are simple to understand, though often their restrictive nature makes them difficult to calculate. It is not necessary to enforce that every player play a single strategy. It is often nice to relax this constraint and recommend that each player pick from a subset of strategies under a probability distribution. This kind of model leads to mixed Nash equilibria.

Definition 2.4. A *mixed Nash equilibrium* is a generalization of a pure Nash equilibrium. It is the result of recommending a probability distribution p_i over S_i to each player i that is independent from the probability distributions given to other players. Let $p_i(s)$ denote the probability that player i should pick strategy $s \in S_i$. Thus, p_i is subject to the following constraints.

$$\sum_{s \in S_i} p_i(s) = 1, \quad \text{and} \quad \forall s \in S_i : \ p_i(s) \geq 0$$

In a two player game, let each player have a probability profile as in Definition 2.4. Assume these profiles are independent from one another and call $|S_1| = m_1$, $|S_2| = m_2$. Then, there is a natural induced probability matrix P that denotes the joint probability of each of the $m_1 \cdot m_2$ outcomes. If $S_1 = \{s_{1,1}, \ldots, s_{1,m_1}\}$ and similarly for S_2, then call entry $P_{i,j} = p_1(s_{1,i}) \cdot p_2(s_{2,j})$ the probability that player one picks strategy $s_{1,i} \in S_1$ and that player two picks $s_{2,j} \in S_2$.

Example 2.1. Let $S_1 = S_2 = \{1, 2, 3\}$ and $p_1(S_1) = p_2(S_2) = \langle \frac{1}{3}, \frac{2}{9}, \frac{4}{9} \rangle$ be the probability distribution for both players one and two. Each player is recommended to pick strategy one $\frac{1}{3}$ of the time, strategy two $\frac{2}{9}$ of the time, and strategy three $\frac{4}{9}$ of the time. Then we have the product matrix

$$P = \begin{pmatrix} 1/9 & 2/27 & 4/27 \\ 2/27 & 4/81 & 8/81 \\ 4/27 & 8/81 & 16/81 \end{pmatrix}.$$

This two player joint probability matrix can be generalized to any number of players. With n players and a recommendation profile p_i for each player i, we have $P(s_{1,j_1}, s_{2,j_2}, \ldots, s_{n,j_n}) = \prod_{i=1}^{n} p_i(s_{i,j_i})$ as the joint probability that player one picks strategy s_{1,j_1}, player two picks strategy s_{2,j_2}, etc.

Like pure Nash equilibria, mixed Nash equilibria have a self-enforcing constraint. For mixed, it is essentially a condition on the expected payoff resulting from switching strategies. For every nonzero probability $p_i(s)$ for player i and under the assumption that all other player's follow their probability recommendations, player i cannot improve his expected utility by switching strategies from s. A profile of probability recommendations $\hat{p} = (p_1, \ldots, p_n)$ is self-enforcing if

$$\forall i \in [1,n], \forall s_i, s_i' \in S_i: \quad p_i(s_i) > 0 \Rightarrow \sum_{\hat{s}_{-i} \in S_{-i}} \left[u_i(\hat{s}_{-i}, s_i) - u_i(\hat{s}_{-i}, s_i')\right] P(\hat{s}_{-i}) \geq 0. \tag{2.2}$$

Even though less restrictive to calculate than pure Nash equilibria, mixed Nash equilibria can still be hard. An even more general concept than both of these concepts is correlated equilibria.

Definition 2.5. Given a game $G = (n, S, U)$, a *correlated equilibrium* is determined by a probability distribution P on S that is used by some external agent in recommending strategy vectors to the players. P is not necessarily a product distribution (i.e., the players need not act independently since there is an external agent involved). It is not in any of the player's best interest to deviate from the recommendation in a similar sense as a mixed Nash equilibrium (Equation 2.2).

Correlated equilibria also have the self-enforcing property. The difference is that the P probability matrix need not come from a product distribution as in a mixed Nash equilibria. Though, it can be reasoned that both mixed and pure Nash equilibria fall under the umbrella of correlated equilibria. Although, some correlated equilibria are outside the realm of Nash equilibria, thus making correlated equilibria a natural generalization of Nash. For correlated equilibria, we have the self-enforcing constraint as

$$\forall i \in [1,n], \forall s_i, s_i' \in S_i: \quad \sum_{\hat{s}_{-i} \in S_{-i}} \left[u_i(\hat{s}_{-i}, s_i) - u_i(\hat{s}_{-i}, s_i')\right] P(\hat{s}_{-i}, s_i) \geq 0. \tag{2.3}$$

3 Discretized Intersection Problem

Traffic systems, as with many physical systems, are often modeled using continuous functions. This is quite natural, as the positions, velocity, accelerations, etc. of the vehicles in the systems do not jump between largely-separated values instantaneously. Even so, it can be instructive to begin discretizing our idea of traffic. Doing so allows us to analyze the system from the perspective of game theory since we treat the vehicles' physical parameters as being chosen from a set of finite choices. Consider the following (slightly ambiguous, but formalized throughout the section) description of the Discretized Intersection Problem (DIP).

Discretized Intersection Problem We are given a grid of size $2r \times 2r$ where the union of the $(r-1)$th and rth rows and columns represent two-way single lane roads meeting at an intersection. Starting at time 0, we choose whether or not to inject cars at four cells $(1, r-1)$, $(r, 1)$, $(2r, r)$, and $(r-1, 2r)$. Notice that this assumes cars drive on the right side of the road. Now at time 1 (time is discrete as well), each car decides whether it would like to move or not, and more cars possibly enter the system. Continuing in this manner and assuming some central device can communicate with the vehicles, can we efficiently route cars through the intersection?

Take as an example the grid system in Figure 1. We will treat this grid as a vehicle intersection with discrete positions, where the valid positions are marked with unit squares. In this intersection, we can allow a car to occupy any valid position $p = (r, c)$ where r is the row and c is the column of the position. As well, a car will have an intended direction $d = (r, c)$ which determines the next adjacent square it would like to move to.

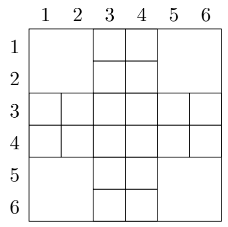

Figure 1: An intersection with discrete positions.

See Figure 2 for an example placement of players. This placement represents a snapshot of our traffic system at a certain discrete time T. What happens when the system advances to time $T + 1$? If all of the cars move in their intended direction, then there will be crash at position $(3, 3)$. In an intuitive sense, this should not be an allowable situation.

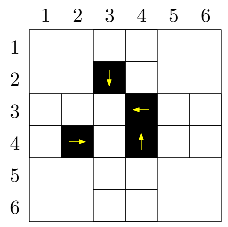

Figure 2: A placement of four cars with intended directions into the intersection.

Instead, we would rather see the traffic configuration advance to the state shown in Figure 3. When routing cars through an intersection, what kind of metrics should determine whether a routing assignment is good? In essence, a driver simultaneously does not want to have an accident and wants to avoid waiting to cross through the intersection.

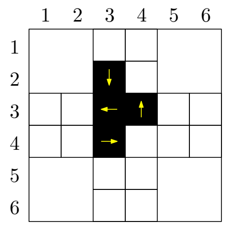

Figure 3: A valid advancement from the state shown in Figure 2.

This brings us motivation to use a game-theoretic description of this system. Let there be n drivers each with a label in $D = [1, n]$. Define the i^{th} driver as a 3-tuple $d_i = (p_i, v_i, w_i)$. p_i is the driver's position in the intersection. v_i is the driver's intended direction represented as a vector (e.g. $(v_i = (1, 0)$ means that driver i intends to move down one row on the next step). w_i counts the number of times that the driver has decided to wait before proceeding forward.

Consider a game $G = (n, \{0,1\}^n, C)$ where $\{0,1\}$ is the set of strategies that any driver may choose from when moving from time T to time $T+1$. Here, 0 is defined to mean that the driver decides to stay in its current position and 1 is defined as moving forward in its intended direction. C is a set of cost functions c_i for each player $1 \leq i \leq n$ that is a map $c_i : \{0,1\}^n \to \mathbb{R}$ satisfying

$$c_i(\hat{s}) = \begin{cases} \infty, & \hat{s} \text{ results in a crash involving } i \\ (1 - s_i)(w_i + 1), & \text{otherwise} \end{cases} \tag{3.1}$$

Upon inspection, this function encompasses both the idea that crashes are undesirable by the drivers yet moving through the intersection with as few delays as possible is desirable. Checking the existence of a crash is possible in $O(n^2)$ time, at least in a naive way wherein for every pair of drivers $d_i = (p_i, v_i, t_i)$ and $d_j = (p_j, v_j, t_j)$, we check if $p_i = p_j$. If true, there is a crash.

We can search for an equilibrium for G which minimizes the sum of the individual costs. This action would take place for each time step, where the new game formed at time T is dependent upon the decisions made during time $T - 1$.

4 Complexity of Naive Solution

The simplest method of finding a correlated equilibrium can be attained by a linear program which simply enumerates the constraints specified by Definition 2.5. This linear program will have size $\Theta(n2^n)$ since each vehicle will have an exponential number of constraints to satisfy as seen in the definition. This is far from efficient, even for relatively small values of n. It seems that we'll need to do a bit more work to achieve an efficient solution.

5 Improvement upon Naive Solution

In order to move towards a more efficient solution to DIP, let us consider a few nice properties that arise as a result of the problem statement.

Crash Locality Notice that the value of each player's cost function is dependent on whether or not the inputted strategy profile (vector of stop / go decisions for each vehicle) results in a crash. When considering if two vehicles i and j could crash on the next time step, are there any conditions on i and j? If not, then we are out of luck in improving cost calculations. Fortunately there is indeed a nice structural property that is necessary for a crash. Namely, if (x_i, y_i) is vehicle i's current position and similarly (x_j, y_j) for vehicle j, then we must have

$$|x_i - x_j| \leq 1 \quad \text{and} \quad |y_i - y_j| \leq 1.$$

Since each player can move at most one grid cell at one time step, two cars can only crash if they are at least one grid cell away from each other. Thus, consider the *dependency graph* in Figure 4. In this graph, there is an edge between a node at position p_i and another at position p_j if and only if there exists a configuration for a vehicle i at p_i and another j at p_j (moving in directions consistent with U.S. traffic laws) such that it is possible for i and j to crash after the next time step.

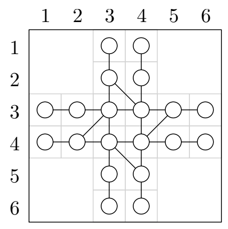

Figure 4: A dependency graph for the grid given in Figure 1.

This graph will prove to be essential for the upcoming connected components discussion, so let's look at a few examples of the definition.

As in the figure, let there be a vehicle u at position $(4, 2)$ moving right and another vehicle v at position $(3, 3)$ moving down. u and v have an edge between them in the dependency graph, so can they collide on the next time step? The answer is yes, if both u and v decide to move forward in their respective directions. This can be thought of as a side-impact collision.

Next example. Let there be a vehicle u at position $(1, 4)$ and another v at $(2, 4)$ both moving downwards. They can both crash on the next step if u decides to move and v decides to stop. In this case, we have a rear-end collision.

Connected Components We have seen that the naive solution builds a linear program of size $\Theta(n2^n)$, but motivated by the dependency graph given above we can make a reasonably effective improvement.

For a given grid and placement of vehicles, we can construct of subgraph of the general dependency graph wherein there is a vertex at position p if and only if there is a driver at position v. Further, there is an edge between u and v if only if they can crash on the next step. We call this graph the *minimal dependency graph*.

Given a minimal dependency graph, the claim is that we can independently solve each connected component as its own instance of DIP, then concatenate the subresults. In fact, this is essentially Theorem 5.1. First, we prove a smaller result.

Fix a strategy profile $\hat{s}$ and two vehicles i and j. Further, let $\delta_k(\hat{s})$ denote the strategy profile which results from changing the kth entry of $\hat{s}$. We have

Claim 5.1. *If i and j are part of different connected components in their minimal dependency graph, then $c_i(\hat{s}) = c_i(\delta_j(\hat{s}))$ and $c_j(\hat{s}) = c_j(\delta_i(\hat{s}))$.*

Proof. Suppose that i and j are two vehicles in different connected components. It follows that there is no edge between i and j. Assume that $c_i(\hat{s}) \neq c_i(\delta_j(\hat{s}))$ or $c_j(\hat{s}) \neq c_j(\delta_i(\hat{s}'))$. By definition of the cost function, this means that either $\delta_j(\hat{s})$ or $\delta_i(\hat{s})$ changed the crash status between i and j. But this is a contradiction, since there is no edge between i and j and we defined an edge in the minimal dependency graph to represent the possibility of a crash. So the claim holds. □

This claim immediately implies the very useful result

Lemma 5.1. *We are given a vehicle i and a strategy vector $\hat{s}$. If $\hat{s}_i$ is the strategy vector that involves only the choices of the vehicles in i's connected component, then $c_i(\hat{s}) = c_i(\hat{s}_i)$.*

Proof. By Claim 5.1, only the decisions of vehicles within i's connected component effect its cost function. So the lemma is true. □

Knowing this lemma will prove to be essential in justifying Theorem 5.1, the main theorem of this section, but hold your breath – we look at a few more pieces of notation and another lemma that will be useful in succinctly maneuvering the theorem.

For two distributions e_1 and e_2 representing recommendation probabilities for the strategies of disjoint sets of vehicles V_1 and V_2, let e_1e_2 be the concatenation of e_1 and e_2 into a joint probability distribution for $V_1 \bigcup V_2$. It will be defined in the usual sense that if $\hat{s}$ is some strategy vector for all vehicles in $V_1 \bigcup V_2$, $\hat{s}$ can be split into two separate vectors $\hat{s_1}$ over V_1's choices and $\hat{s_2}$ over V_2's choices. Then, $e_1e_2(\hat{s}) = e_1(\hat{s_1})e_2(\hat{s_2})$.

Notice that the connected components of an undirected graph G define equivalence classes on G. So given a minimal dependency graph D for DIP, we can express D as the disjoint union $D_1 \bigcup D_2 \bigcup \ldots \bigcup D_k$ where each D_i is a connected component of D and $|D_i| = n_i$. Write opt_i as the optimal correlated equilibrium for D_i (with the global cost as sum of the individual costs).

Lemma 5.2. *$P = \prod_{i=1}^{k} opt_i$ defines a correlated equilibrium for D.*

Proof. In order to show P is a correlated equilibrium for D, we need to verify that a vehicle cannot not deviate from the distribution specified by P to improve its cost. Checking that the following cases do not occur is sufficient.

1. **P recommends a crash.** Since a crash results in infinite cost, an optimal equilibrium will never choose to include such a strategy (stopping all cars has a smaller cost). But each potential crash is contained in each connected component and every opt_i is optimal, so this case does not occur.
2. **P favors some car stopping when it could instead move safely.** If a car could move forward safely, that means that it could move forward without a crash thus improving its cost. But some opt_i would recognize that this crash would not occur, since all crash information is contained in each connected component. Since opt_i is optimal, it will have already moved cars that can do so safely. So this case does not occur, either.

Therefore, P is a correlated equilibrium. □

Finally, we are ready to prove the main theorem!

Theorem 5.1. *$opt = \prod_{i=1}^{k} opt_i$ is an optimal correlated equilibrium.*

Proof. We have already seen in Lemma 5.2 that the product of the opt_i's is a correlated equilibrium. We now show it is optimal. Note that, by definition, each opt_i results in a minimal expected global cost within D_i. We have

$$\zeta_i = \sum_{j=1}^{n_i} \sum_{\hat{s} \in \{0,1\}^{n_i}} c_j(\hat{s}) opt_i(\hat{s})$$

is minimal. Also, we will write opt_{-i} to mean $\prod_{j=1, j\neq i}^{k} opt_j$ which implies the nice expression $opt = (opt_i)(opt_{-i})$. We'll abuse notation so that for a vehicle j, opt_j will be the appropriate correlated equilibrium for j's connected component. Observe that the global expected cost for the product distribution is

$$\begin{aligned}
&\sum_{j=1}^{n} \sum_{\hat{s} \in \{0,1\}^n} c_j(\hat{s}) opt(\hat{s}) && (5.1)\\
=&\sum_{j=1}^{n} \sum_{(\hat{s}_j, \hat{s}_{-j}) \in \{0,1\}^n} c_j(\hat{s}_j, \hat{s}_{-j}) opt_j(\hat{s}_j) opt_{-j}(\hat{s}_{-j}) && (5.2)\\
=&\sum_{j=1}^{n} \sum_{(\hat{s}_j, \hat{s}_{-j}) \in \{0,1\}^n} c_j(\hat{s}_j) opt_j(\hat{s}_j) opt_{-j}(\hat{s}_{-j}) && (5.3)\\
=&\sum_{j=1}^{n} \left(\sum_{\hat{s}_j \in \{0,1\}^{n_j}} c_j(\hat{s}_j) opt_j(\hat{s}_j) \right) \left(\sum_{\hat{s}_{-j} \in \{0,1\}^{n-n_j}} opt_{-j}(\hat{s}_{-j}) \right) && (5.4)\\
=&\sum_{j=1}^{n} \sum_{\hat{s}_j \in \{0,1\}^{n_j}} c_j(\hat{s}_j) opt_j(\hat{s}_j) && (5.5)\\
=&\sum_{i=1}^{k} \zeta_i && (5.6)
\end{aligned}$$

(5.3) is true due to Lemma 5.1, and (5.5) is true since the right hand sum in (5.4) ranges over all of the probabilities in the distribution opt_{-j}. Since each ζ_i is minimal and can only take on nonnegative values (due to individual cost function being nonnegative), (5.6) is also minimal. The theorem is proved. □

As promised in the beginning of the section, we now have a means to say that we can construct a better linear program in some cases. In fact, by the theorem we see that we can create k different linear programs and solve each of them independently to achieve the optimal correlated equilibrium. In the best case, we solve n linear programs each of size $\Theta(1)$ when each connected component consists of a single vehicle. Though in the worst case, we are back in the previous situation with a linear program of size $\Theta(n2^n)$. Still, this is some improvement.

6 Implication Graph

With the properties exploited in the previous section, we now will discuss a promising idea for constructing a polynomial-sized linear program for DIP. The algorithm will be introduced, though no proof of correctness will be given at this time.

Notice that a correlated equilibrium will **never** choose a nonzero probability for recommending a strategy profile which dictates a crash to occur. A crash between two vehicles results in an infinite individual cost for those two vehicles and resultingly an infinite global cost. The correlated equilibrium could trivially find an equilibrium with smaller global cost by telling all vehicles to stop. Additionally, many of the strategies vectors which we consider contain crashes, so perhaps we could prune these vectors out of our linear program.

It also helps to notice that if a car u is trailing another car v and if v is not in the intersection, then u will always benefit from moving if v decides to move. What we see here is that we have some decision-based implications between players. We saw in the last section what was referred to as the dependency graph of an intersection, but not consider the interesting *implication graph.*

The implication graph of an instance of DIP will be a directed – possibly cyclic – graph representing decision-induced dependencies between vehicles. There will be two vertices for each vehicle i: x_i which represents the decision for i to move and the other $\bar{x}_i$ for i to stop. Then there will be a directed edge between nodes x and x' when the decision dictated by x requires the decision dictated by x' (otherwise a crash). Since we saw in the previous section that each connected component can be solved independently, we can assume that this implication graph is weakly connected.

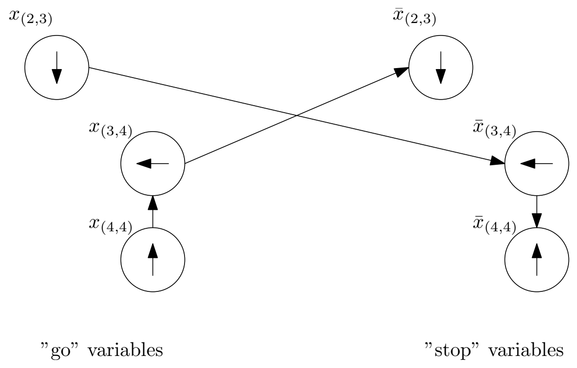

Figure 5: An implication graph for the three vehicle connected component from Figure 2.

This implication graph is reminiscent of the skew-symmetric graphs created when solving 2-SAT instances with the method of Aspvall, Plass & Tarjan [1]. Our algorithm will be inspired by this, but since 2-SAT is only concerned with finding a single satisfying assignment and here we wish to find a specific set of assignments, there will be some major differences. Without further ado, here is the outline for the algorithm.

1. Create implication graph as above.

2. Condense the implication graph by treating each strongly-connected component as a vertex itself. The resulting graph will be acyclic.

3. For each vertex v in the condensed graph with indegree 1, do a breadth first search rooted at v to find out if it implies any contradictions. v implies a contradiction if

its BFS tree includes both x_i and $\bar{x}_i$ for some i. If so, then delete all the variables in the tree which imply the contradiction, and all edges coming out of them as well.

4. For each vertex v with indegree 1 in the modified condensed graph, there will be a distinct assignment of decisions. Find all of these.

5. From the sets of decisions found in the previous step, use these as the valid strategy profiles for computing a correlated equilibria in the DIP instance.

Concretely, we see in Figure 5 that there are two strategy profiles to consider. One is where the vehicles at $(3,4)$ and $(4,4)$ move and the vehicle at $(2,3)$ waits. The other is where the vehicle at $(2,3)$ moves and the others wait.

The claim here is that there will be at most four nodes with indegree of 1 in the implication graph of each subinstance of DIP. Therefore, we have can create a linear program over these strategy profiles to find the optimal corrrelated equilibrium with size $\Theta(n)$. This is true for each connected component in the dependency graph, so for k connected components we have k linearly sized linear programs.

7 Future Work

Through this paper, we have seen that it is possible to efficiently solve DIP at each time step. But, there are many questions left unanswered.

- Does finding an optimal correlated equilibrium for each time step perform optimally from a global perspective? In other words, is it safe to assume that performing optimally at time T without regard to the future does not negatively effect future cost evaluations?

- Can we allow more than two choices for each vehicle? In other words, can we more realistically model the speeds of a car besides a binary 'stop' and 'go' while still maintaining efficiency?

- Can we expand the model to incorporate large road networks, which would essentially be joining together multiple intersections?

These questions will serve as a basis for continuing this research next semester.

References

[1] Bengt Aspvall, Michael F., and Robert E. Tarjan. “A linear-time algorithm for testing the truth of certain quantified boolean formulas”. In: *Information Processing Letters* (1979), pp. 121–123.

[2] Noam Nisan et al. *Algorithmic Game Theory.* Cambridge University Press, 2007.

[3] Christos H. Papadimitriou and Tim Roughgarden. “Computing correlated equilibria in multi-player games”. In: *Journal of the ACM* 55.14 (3 July 2008).