# Algorithmic Game Theory

Final Report for CMSC451 Honors Option

Robert Adkins
Fall 2015

Faculty Advisor: David Mount

**Abstract**

In this paper, I introduce some fundamental concepts to the field of algorithmic game theory. I also explore some of the known algorithms for calculating various kinds of equilibria in games, like the Lemke-Howson algorithm for finding mixed Nash equilibria and the Ellipsoid against Hope algorithm for finding correlated equilibria. After these ideas are introduced, potential applications towards autonomous vehicle traffic routing are explored. This latter section yields the motivation behind my study of algorithmic game theory over the course of this semester.

---

# Contents

# 1 Introduction

I began studying algorithmic game theory as an augmentation to CMSC451 with Dr. David Mount. The work done throughout this semester will carry through to next semester under the intention to produce original results. This section will address some of the fundamental concepts of game theory that I have studied this semester. The definitions placed in this section will be referenced in later sections.

## 1.1 Algorithmic Game Theory

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 1.1.** 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.

Algorithmic game theory is concerned with the computability of games. Is it feasible to represent a game with many players on a computer? This question is explored in Section 2. If so, can we efficiently calculate the behavior of the game over time (Section 3)? These questions – among others – have been major sources of motivation behind recent research in algorithmic game theory.

## 1.2 Equilibria Concepts

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 1.2.** We are given a game $G = (n, S, U)$ as defined in Definition 1.1. 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. The most basic form of equilibria is a pure Nash equilibria.

**Definition 1.3.** A *pure Nash equilibrium* is an equilibrium that occurs when players are recommended to deterministically play a single strategy all of the time.

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 $i$th 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 $\hat{s} = (\hat{s}_{-i}, s_i) \in S$,

$$\forall i \in [1, n], \forall s_i' \in S_i : \quad u_i(\hat{s}_{-i}, s_i) \geq u_i(\hat{s}_{-i}, s_i'). \tag{1.1}$$

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 oftentimes 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 1.4.** 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 1.4. 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 1.1.** 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

$$\forall i \in [1, n], \forall s_i, s_i' \in S_i : \quad p_i(s_i) > 0 \Rightarrow \sum_{\hat{s}_{-i} \in S_{-i}} \left[ u_i(\hat{s}_{-i}, s_i) - u_i(\hat{s}_{-i}, s_i') \right] P(\hat{s}_{-i}) \geq 0. \tag{1.2}$$

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 1.5.** 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 1.2).

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

$$\forall i \in [1,n], \forall s_i, s_i' \in S_i : \quad \sum_{\hat{s}_{-i} \in S_{-i}} \left[u_i(\hat{s}_{-i}, s_i) - u_i(\hat{s}_{-i}, s_i')\right] P(\hat{s}_{-i}, s_i) \geq 0. \tag{1.3}$$

Moving away from the previous ideas, the following equilibria concept employs a new model of player behavior. A player's *regret* is defined as the difference between the average utility of the player's strategy choices and the average utility of the best choices that could have occurred. This form of behavior was proposed in [1].

**Definition 1.6.** In *regret minimization* all players, instead of worrying about switching strategies from only their current strategy, look at how much they would regret switching strategies when considering **all** past decisions.

# 2 Representation of Games

If we wish to utilize games to compute solutions to problems, then we must have some method to concretely represent the games. This section explores various ways in which to represent games, both generally and in special cases.

## 2.1 Exponential Representation of Games

The naive approach toward representing games is just to list every piece of information. In a general game this means that there is an exponential amount of information.

**Claim 2.1.** *Given a game $G = (n, S, U)$ as in Definition 1.1, call $\max(|S_i|) = s$. Then, $G$ requires $O(s^n)$ space.*

*Proof.* Suppose we have a game $G = (n, S, U)$ and with $s$ as above. Without loss of generality assume that $S_1 = S_2 = \ldots = S_n$ and that $|S_i| = s$. Then, each player has a choice of $s$ strategies. The number of permutations with repetition for $n$ players to pick $s$ strategies is $s^n$. Additionally, each player $i$ will have a unique payoff for each of these $s^n$ situations given by the utility function $u_i \in U$. $u_i$ can be represented by an $n-dimensional$ matrix with size $s$. Since each player has one of these matrices, there is $ns^n$ total information. □

This claim hints toward the computational hardness of analyzing general games. Since there is such a large upper bound on the amount storage, it naturally follows that designing efficient algorithms for games with many players is a difficult task. For this reason, we often restrict our attention to 2-player games. A discussion of the computational complexity of determining equilibria in games can be found in Section 3.1.

### 2.2 Polynomial Representation of Succinct Games

The previous section may have disheartened some of you, but do not despair! We can create polynomial representations of certain games if we consider adding restrictions to general games. I will list and describe a few of these succinctly representable games. In these games, take $G$ and $s$ as in the proof of Claim 2.1.

**Symmetric Games** This is the oldest class of succinctly representable games. In symmetric games, there are no distinguishing factors between players. Because of this, the number of distinguishable outcomes is drastically diminished from that of Claim 2.1. Instead of considering the number of choices each player has, we now care about how to distribute $n$ players among $s$ strategies (combinations with repetition). This is precisely $\binom{n+s-1}{n}$. When $s << n$, this requires $O(n^s)$ space – a drastic improvement from $O(s^n)$.

Symmetric games are well-studied, though they are restricted in the sense that we cannot distinguish the actions of individual players.

**Graphical Games** Graphical games are games in which each player's utility does not necessarily depend on all other players' choices. Instead, the utility of a single player is only dependent on a small subset of the other players. With this notion, one can construct a dependency graph in which each node represents a player, and there is an edge from player $i$ to player $j$ if and only if $i$'s strategy depends on $j$'s choice of strategy.

In graphical games, one can calculate correlated equilibria (Definition 1.5) in polynomial time if the associated dependency graph has bounded tree-width. This limited result was proved by Papadimitriou and Roughgarden in [4].

**Sparse Games** Sparse games are named in reference to sparse matrices. If the players' utility functions are defined to be zero in most situations, then it suffices to store only the non-zero utilities and their associated strategy vectors. This clearly reduces the amount of information needed to store the game.

Due to the exponential nature of general games, most of the current literature deals with specific succinctly representable games on a case-by-case basis. Doing so allows more meaningful analysis.

## 3 Computing Equilibria

Now that many of the foundational concepts of game theory have been introduced, I will present topics related to the computability of equilibria. It turns out, unsurprisingly, that the computability of equilibria is dependent upon the type of equilibria in question.

**Theorem 3.1.** *(Nash) Every finite game has a mixed Nash equilibrium.*

*Proof.* Omitted. See [3]. □

This theorem was first proved by Nash. By finite game, this means that the set of strategies has finite cardinality as does the set of players. A discrete version of the Brouwer fixed-point theorem can be used to prove Theorem 3.1.

For this section, it will be instructive to use an example game to show calculations. Consider the following Color Game.

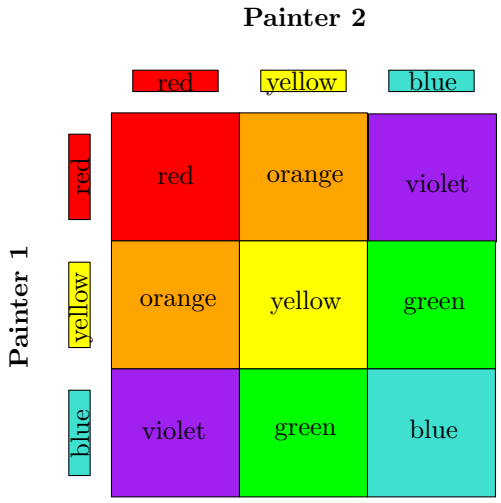



Figure 1: Possible colors resulting from picking two of red, yellow, and blue.

**Color Game.** Two painters are situated on either side of a wall with no means of communication with one another. Each has unlimited access to buckets of red, yellow, and blue paint (each bucket holds one color of paint). Each also has unlimited access to empty buckets. On both sides, there is a funnel leading into the wall and a tube leading out. The painters are each instructed to choose a bucket of paint which they will pour down their respective funnels. After doing so, the painters' choices are mixed together inside the wall and the mixture is dispensed back to both sides through the tubes. The painters will catch the new paint mixture with their empty buckets. As a reference, the following matrix shows the possible outcomes.

There are a total of 9 permutations which describe all of the possible outcomes. One might think that this double counts for duplicate colors, but this is not completely true. Both painters have certain preferences that distinguish some of these duplicates. The painters have identical conditions that are described as follows.

1. Neither wants to receive a mixture of red, yellow, or blue (after all, they already have unlimited supplies of these colors!).
2. Each painter desires violet paint above all other colors, but only if it is made with his own blue (the "better" blue).
3. If a painter wastes his precious blue paint on a green mixture, then he is upset.
4. If one painter notices the other wastes his blue paint on a green mixture, then he is filled with schadenfreude.
5. Either painter will settle for orange.

These five conditions are captured in the following utility matrix for Painter 1 (following same layout as Figure 3).

$$U = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 2 \\ 2 & 0 & 0 \end{pmatrix}$$

Since the painters have identical preferences, $U^\intercal$ is the utility matrix for Painter 2. Define $red := 1$, $yellow := 2$, and $blue := 3$. Additionally, call $u_1(i,j) := U_{i,j}$ and $u_2(i,j) := U^\intercal_{i,j}$. Therefore, we formally defined the Color Game as $G = (2, \{red, yellow, blue\}^2, \{u_1, u_2\})$. We will revisit this game when considering various equilibria.

## 3.1 Complexity

Papadimitriou introduced the concept of *polynomial parity arguments on directed graphs* (**PPAD**), a complexity class that includes computing mixed Nash equilibria [3]. Similar to the class NP, there is an established set of PPAD-complete problems which are believed to be computationally difficult.

One might question why the class PPAD is used instead of NP to represent the intractability of calculating Nash equilibria. The answer is that NP problems are classically a set of decision problems of the form, "Does a solution exist to this problem?". Well, Theorem 3.1 guarantees the existence of a mixed Nash equilibria, so the problem of computing equilibria does not seem to fit well within the framework of NP.

The class PPAD is defined by a reduction of the problem at hand to a directed graph.

**Definition 3.1.** A problem $p$ is in *PPAD* if $p$ is representable as a directed graph $D = (V, E)$ with the following conditions satisfied.

1. $|V| < \infty$ yet is possibly exponentially large.
2. The indegree and outdegree of each vertex in $V$ is at most one.
3. Given a label $L$, it is computationally easy to identify if $L$ is a vertex in $V$ and to determine the predecessor and successor of $L$.
4. There are source vertices of indegree zero (which come paired with sink vertices of outdegree zero).
5. Any sink of $D$ represents a solution to $p$.

It turns out that the concept of PPAD-completeness is weaker than NP-completeness.

## 3.2 Nash Equilibria

The classic algorithm to calculate Nash equilibria is the Lemke-Howson algorithm. It actually constructively suggests that calculating Nash equilibria is PPAD-complete.

### 3.2.1 Lemke-Howson

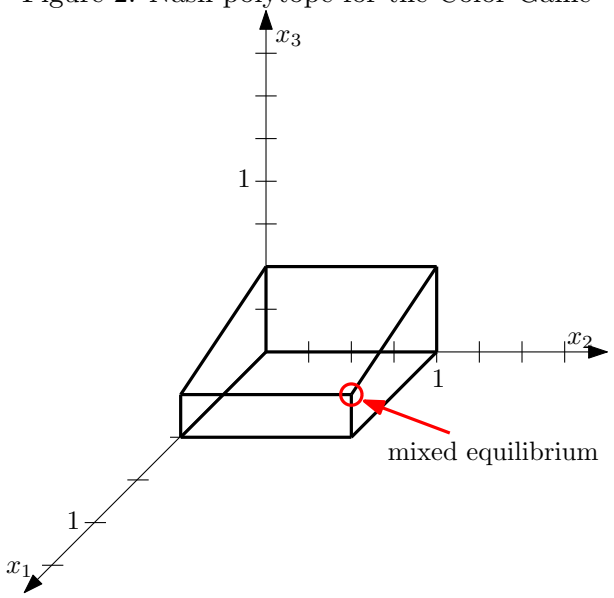
In the case of two player symmetric games, the Lemke-Howson algorithm [3, 5] calculates the Nash equilibria. Given a two player symmetric game $G = (2, S, \{u_1, u_2\})$, there is a matrix $\mathbf{A}$ that represents $u_1$ and by symmetry $\mathbf{A}^\intercal$ represents $u_2$. Then, using the constraints of a Nash equilibrium as specified in Definition 1.4, we have the following matrix inequalities, solving for $\vec{x}$.

$$\mathbf{A}\vec{x} \geq 0$$
$$\mathbf{A}\vec{x} \leq 1$$

These inequalities define a convex polytope (as long as the system is non-degenerate). Each vertex of the polytope will be associated with some of the strategies complying with constraints. Then, one can start at any vertex of the polytope, and traverse the edges until reaching a vertex at which **all** strategies are represented (a strategy $i$ is represented if $x_i = 0$ or $\mathbf{A}_i\vec{x} = 1$). Normalizing this vertex yields a Nash equilibrium.

**Example 3.1.** (Color Game) Consider $G = (2, \{red, yellow, blue\}^2, \{u_1, u_2\})$ as defined early in the section. Then, the Nash polytope associated with the utility matrix $U$ looks as in Figure 2.

Figure 2: Nash polytope for the Color Game




It turns out that the only nonzero vertex of the above polytope which has all strategies represented is the point $(\frac{1}{2}, 1, \frac{1}{4})$. All strategies are represented due to the following equality and the fact that all of the entries of $(\frac{1}{2}, 1, \frac{1}{4})$ are nonzero.

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 2 \\ 2 & 0 & 0 \end{bmatrix} \begin{bmatrix} 1/2 \\ 1 \\ 1/4 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

Normalizing this vector gives the symmetric mixed Nash equilibrium: $\hat{p} = (\frac{2}{7}, \frac{4}{7}, \frac{1}{7})$. The joint probability distribution induced by $\hat{p}$ gives an expected global utility (sum of individual utilities) of $\frac{56}{49}$. Can we do better than this? The answer is yes, if we use a correlated equilibrium!

As a side note, notice that the color game has **no** pure Nash equilibria.

### 3.3 Correlated Equilibria

The algorithm for calculating correlated equilibria was established in [4] and was based upon a previously developed existence proof. The algorithm is cutely termed the Ellipsoid against Hope algorithm and is built upon linear programming duality, the ellipsoid algorithm, and Markov chain steady state computations. In this subsection, I will give a slight digression into linear programming duality then state an overview of the Ellipsoid against Hope algorithm.

### 3.3.1 Linear Programming and Duality

Finding equilibria fits nicely into the framework of linear programming. Consider a symmetric game $G = (n, S, U)$ which has a global utility function defined to be the sum of the player's individual utilities (this is a commonly used social utility function). Additionally, we assume each $u_i \in U$ to be n-linear. Then, we can succinctly represent computing a correlated equilibria as the following linear program.

$$\max \left( \sum_{i=1}^{n} u_i(\vec{x}) \right) \tag{3.1}$$

$$\mathbf{C}\vec{x} \geq \vec{0} \tag{3.2}$$

$$\vec{x} \geq \vec{0} \tag{3.3}$$

$$\vec{x} \cdot \vec{1} = 1 \tag{3.4}$$

Here, Equation 3.1 says to optimize the global utility function. Equation 3.2 encapsulates the constraints of a correlated equilibrium (Equation 1.3). Equations 3.3 and 3.4 ensure that $\vec{x}$ represents a probability distribution (since $\vec{x}$ represents a strategy vector). To move forward from here, consider the following theorem.

**Theorem 3.2.** *Every finite game has a correlated equilibrium.*

*Proof.* Follows directly from Theorem 3.1. A finite game is guaranteed to have a mixed Nash equilibrium. Call this equilibrium $E$. By the definition of correlated equilibrium (1.5), $E$ clearly satisfies all of the constraints. Therefore, $E$ is a correlated equilibrium. □

Since $G$ is guaranteed to have a solution, its linear program is guaranteed to succeed. By duality, the dual program of $G$ is guaranteed to fail. Let's go on a slight tangent to describe some aspects of linear programming duality.

*Remark.* Much of the remaining content in this section is from [2]. In the matrix equations that follow, I will loosely interchange column and row vectors.

A linear program will seek to either minimize or maximize some linear function. Without loss of generality, consider the following maximization-based linear program (called the primal).

$$\begin{aligned} &\max (\vec{b} \cdot \vec{x}) \\ &\mathbf{A}\vec{x} \leq \vec{c} \qquad (P) \\ &\vec{x} \geq \vec{0} \end{aligned}$$

When constructing the dual linear program of P, we desire to create a minimization linear program such that its optimal solution matches the optimal solution of P. One can do so by taking a linear combination of the rows of $A$ in P such that it provides a close upper bound on $b$. Then, the dual linear program is this.

$$\begin{aligned} &\min (\vec{c} \cdot \vec{y}) \\ &\mathbf{A}^{\intercal}\vec{y} \geq \vec{b} \qquad (D) \\ &\vec{y} \geq \vec{0} \end{aligned}$$

### 3.3.2 Ellipsoid against Hope

Take Equations 3.1 - 3.4 in the previous section as the primal program (P). As stated before, (P) is guaranteed to be feasible which implies that its dual (D) will be infeasible. Thus, the ellipsoid algorithm (used to solve linear programs) on (D) will eventually halt and state that (D) is infeasible. But, the ellipsoid algorithm will collect information at each iteration that can be used to construct a feasible solution in (P). This is the correlated equilibrium. More details on this are in [4].

**Example 3.2.** (Color Game) Looking back at the Color Game, we previously found a Nash equilibria $\hat{p} = (\frac{2}{7}, \frac{4}{7}, \frac{1}{7})$ that gives an expected global utility of $\frac{56}{49} = \frac{8}{7}$. Applying linear programming directly to the correlated equilibria self-enforcing condition given in Equation 1.3 yields the following distribution (See the attached GNU MathProg model used to calculate this result).

$$P = \begin{pmatrix} 0 & 1/3 & 1/6 \\ 1/6 & 0 & 1/12 \\ 1/12 & 1/6 & 0 \end{pmatrix}$$

As would be expected from a general correlated equilibrium, $P$ is not a product distribution and is therefore not a Nash equilibria. To see this notice that the $rank(P) \neq 1$.

Remarkably, $P$ gives an expected global utility of 2 which is much better than $\frac{8}{7}$! But, added global utility seems to come at a cost. Notice that $P$ is not symmetric, so $P$ does not evenly distribute the global utility among the individual utilities. In fact Painter 1 has an expected utility of $\frac{5}{6}$ while Painter 2 has one of $\frac{7}{6}$.

# 4 Optimality Metrics

It is important to be able to rationalize the optimality of an equilibrium solution in a game. In order to do so, mathematicians have long used the *price of anarchy*. Though, more recent is the notion of the *price of total anarchy*. In both definitions, we wish the value to be as close to 1 as possible. Both of the notions are explained below.

## 4.1 Price of Anarchy

**Definition 4.1.** The *price of anarchy* of a game is defined to be the ratio of the optimal global utility to the worst global utility of an equilibrium solution.

**Example 4.1.** (Color Game) Consider the Color Game $G = (2, \{red, yellow, blue\}^2, \{u_1, u_2\})$ from Section 3. Example 3.1 showed that $\hat{p} = (\frac{2}{7}, \frac{4}{7}, \frac{1}{7})$ is the only mixed Nash equilibrium and has expected sum of individual utilities $\frac{8}{7}$. Considering that $u_1 + u_2$ is captured by the matrix

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 2 \\ 2 & 0 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 1 & 2 \\ 1 & 0 & 0 \\ 0 & 2 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 2 & 2 \\ 2 & 0 & 2 \\ 2 & 2 & 0 \end{bmatrix},$$

it is clear that $\max(u_1 + u_2) = 2$. So, the price of anarchy for using mixed Nash equilibria is $\frac{7}{4}$.

### 4.2 Price of Total Anarchy

The price of total anarchy was developed in [1].

**Definition 4.2.** The *price of total anarchy* of a game is the ratio of the optimal global utility to the global utility incurred when players are trying to minimize regret.

## 5 Applications to Autonomous Vehicle Traffic Routing

There is a rapid emergence of new technologies that are providing automated driving assistance in vehicles. It is not difficult to imagine a world in which these currently isolated features are connected into a cohesive system which utilizes inter-vehicle communication [6]. Doing so could enable vehicles to coordinate their motion in order to improve traffic flow. The effectiveness of this coordination will depend on the algorithms that are developed to accomplish it. This motivates research into the design of efficient algorithms that will coordinate the routing of automated vehicle systems.

The question comes to mind: What is an efficient algorithm to optimally route autonomous vehicles? I will henceforth refer to this question as the autonomous vehicle traffic routing (AVTR) problem. As explained above, AVTR is of great importance to society and to the algorithms community, motivating my aim to conduct research in the pursuit of its solution. In order to approach AVTR, it is necessary to develop an appropriate model for the traffic system. There are a number of methods that have been proposed to model traffic, including physically-influenced models based upon partial differential equations, configuration-space approaches from the field of robotics, and optimal network routing from the field of operations research. I propose to study a model of autonomous vehicle traffic based on a combination of algorithmic game theory and network flow. Over the course of this semester, I have been preparing with Dr. David Mount to conduct research into an algorithm resulting from this model.

One of my ideas is to model the behavior of the dynamic network flow as a *graphical game*, meaning that an individual flow's routing decision is influenced by a small subset of the other flows. Using a small subset allows polynomial time computation of an optimal correlated equilibrium if the graph representation of the game is of bounded tree-width [4]. In the context of AVTR, a correlated equilibrium can be interpreted as a route recommended to each flow by some coordinator who has knowledge of the subset of flows that influence the flow in question. I propose that a flow's dependent subset be determined by designing its utility function to account for potential overlaps between its path and other flows' paths. Introducing this slight interdependency among utility functions could lead to more socially optimal routing than is seen with the selfish driver model.

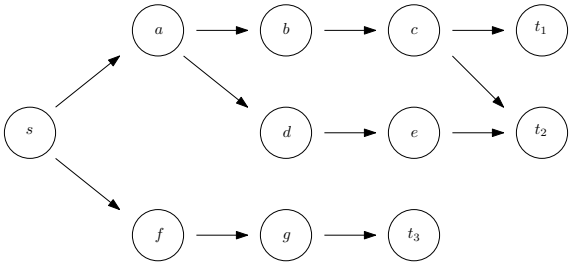



Figure 3: A multi-sink network flow graph

Consider Figure 3 (the edge weights are omitted for simplicity). All flows start from the source $s$ and end at one of the sinks $t_1$, $t_2$, or $t_3$. Call $F_i$ the class of flows that end at $t_i$. Then, it follows that a flow $f_1 \in F_1$ and a flow $f_2 \in F_2$ could potentially overlap (e.g., $f_1$ travels the path $\langle s, a, b, c, t_1 \rangle$ and $f_2$ travels the path $\langle s, a, b, c, t_2 \rangle$). In contrast, $F_3$ does not overlap with either $F_1$ or $F_2$ except trivially at $s$. These dependencies motivate the graphical game wherein the vertices are $F_1$, $F_2$, and $F_3$, and the directed edges are $(F_1, F_2)$ and $(F_2, F_1)$. This graph encompasses the idea that a coordinator may want to be careful when recommending routes to flows in $F_1$ and $F_2$ since the routing of a flow in one class could influence the travel time of a flow in the other class due to overlap. For example, a coordinator would want to route a fraction of the $f_2 \in F_2$ on the path $\langle s, a, d, e, t_2 \rangle$ to avoid some overlap with $F_1$ flows. The main point in this example is that creating a dependency graph for a general network flow may allow the game's correlated equilibrium to produce socially optimal routing.

# 6 Future Work

I plan to use the results from this document as a basis in researching efficient algorithms for autonomous vehicle traffic routing. It is possible that the research may involve an approach which is not exactly as presented in Section 5. For example, there may be more of a focus on regret minimization than correlated equilibria. As well, the problem of efficiently routing autonomous vehicle traffic may turn out to be intractable, thus moving the research in the direction of an approximation algorithm.

# References


[1] Avrim Blum et al. "Regret Minimization and the Price of Total Anarchy". In: *Proceedings of the Fortieth Annual ACM Symposium on Theory of Computing*. Victoria, British Columbia, Canada: ACM, 2008, pp. 373–382.

[2] Samir Khuller. "CMSC651 Lecture Notes (19, 20, 22)". `https://www.cs.umd.edu/class/spring2011/cmsc651/lecs.html`. 2011.

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

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

[5] LloydS. Shapley. "A note on the Lemke-Howson algorithm". English. In: *Pivoting and Extension*. Ed. by M.L. Balinski. Vol. 1. Mathematical Programming Studies. Springer Berlin Heidelberg, 1974, pp. 175–189. ISBN: 978-3-642-00756-9. DOI: 10.1007/BFb0121248. URL: http://dx.doi.org/10.1007/BFb0121248.

[6] M.L. Sichitiu and M. Kihl. "Inter-vehicle communication systems: A survey". In: *Communications Surveys Tutorials, IEEE* 10.2 (2008), pp. 88–105.