ALGAMES: A Fast Solver for Constrained Dynamic Games

Simon Le Cleac'h
Department of Mechanical Engineering
Stanford University
simonlc@stanford.edu

Mac Schwager
Department of Aeronautics
& Astronautics
Stanford University
schwager@stanford.edu

Zachary Manchester
Department of Aeronautics
& Astronautics
Stanford University
zacmanchester@stanford.edu

***Abstract*—Dynamic games are an effective paradigm for dealing with the control of multiple interacting actors. This paper introduces ALGAMES (Augmented Lagrangian GAME-theoretic Solver), a solver that handles trajectory optimization problems with multiple actors and general nonlinear state and input constraints. Its novelty resides in satisfying the first order optimality conditions with a quasi-Newton root-finding algorithm and rigorously enforcing constraints using an augmented Lagrangian formulation. We evaluate our solver in the context of autonomous driving on scenarios with a strong level of interactions between the vehicles. We assess the robustness of the solver using Monte Carlo simulations. It is able to reliably solve complex problems like ramp merging with three vehicles three times faster than a state-of-the-art DDP-based approach. A model predictive control (MPC) implementation of the algorithm, running at more than 60Hz, demonstrates ALGAMES' ability to mitigate the "frozen robot" problem on complex autonomous driving scenarios like merging onto a crowded highway.**

I. Introduction

Controlling a robot in an environment where it interacts with other agents is a complex task. Traditional approaches in the literature adopt a predict-then-plan architecture. First, predictions of other agents' trajectories are computed, then they are fed into a planner that considers them as immutable obstacles. This approach is limiting because the effect of the robot's trajectory on the other agents is ignored. Moreover, it can lead to the "frozen robot" problem that arises when the planner finds that all paths to the goal are unsafe [1]. It is, therefore, crucial for a robot to *simultaneously* predict the trajectories of other vehicles on the road while planning its own trajectory, in order to capture the reactive nature of all the agents in the scene. ALGAMES provides such a joint trajectory predictor and planner by considering all agents as players in a Nash-style dynamic game. We envision ALGAMES being run on-line by a robot in a receding-horizon loop, at each iteration planning a trajectory for the robot by explicitly accounting for the reactive nature of all agents in its vicinity.

Joint trajectory prediction and planning in scenarios with multiple interacting agents is well-described by a dynamic game. Dealing with the game-theoretic aspect of multi-agent planning problems is a critical issue that has a broad range of applications. For instance, in autonomous driving, ramp merging, lane changing, intersection crossing, and overtaking maneuvers all comprise some degree of game-theoretic interactions [2, 3, 4, 5, 6, 7]. Other potential applications include mobile robots navigating in crowds, like package delivery robots, tour guides, or domestic robots; robots interacting with people in factories, such as mobile robots or fixed-base multi-link manipulators; and competitive settings like drone and car racing [8, 9].

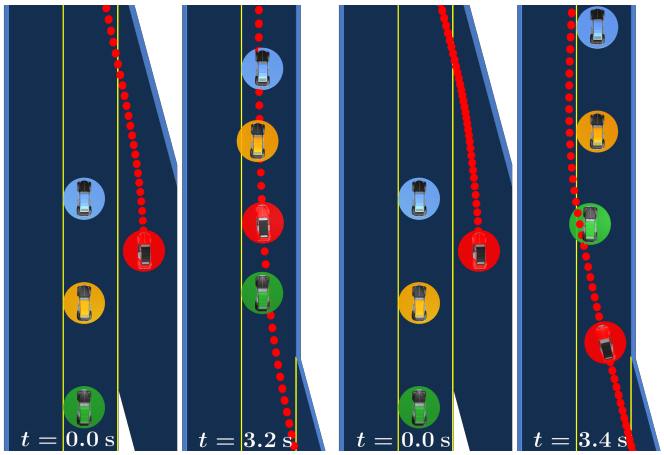

Fig. 1. A merging maneuver on a crowded highway is carried out using a receding horizon implementation of ALGAMES. On the left, the red car controlled by ALGAMES merges between the orange and green cars with minimal disruption to the traffic. On the right, the red car controlled by a non game-theoretic MPC, is "frozen" i.e., it cannot find a feasible path. It has to slow down significantly and wait for the three cars to pass before merging.

In this work, we seek solutions to constrained multi-player general-sum dynamic games. In dynamic games, the players' strategies are sequences of decisions. It is important to notice that, unlike traditional optimization problems, non-cooperative games have no "optimal" solution. Depending on the structure of the game, asymmetry between players, etc., different concepts of solutions are possible. In this work, we search for Nash equilibrium solutions. This type of equilibrium models symmetry between the players; All players are treated equally. At such equilibria, no player can reduce its cost by unilaterally changing its strategy. For extensive details about the game-theory concepts addressed in this paper, we refer readers to the work of Bressan [10] and Basar et al. [11].

Our solver is aimed at finding a Nash equilibrium for multi-player dynamic games, and can handle general nonlinear state and input constraints. This is particularly important for robotic applications, where the agents often interact through their desire to avoid collisions with one another or with the environment. Such interactions are most naturally represented

as (typically nonlinear) state constraints. This is a crucial feature that sets game-theoretic methods for robotics apart from game-theoretic methods in other domains, such as economics, behavioral sciences, and robust control. In these domains, the agent interactions are traditionally represented in the objective functions themselves, and these games typically have no state or input constraints. In mathematics literature, Nash equilibria with constraints are referred to as *Generalized Nash Equilibria* [12]. Hence, in this paper we present an augmented Lagrangian solver for finding Generalized Nash Equilibria specifically tailored to robotics applications.

Our solver assumes that players are rational agents acting to minimize their costs. This rational behavior is formulated using the first-order necessary conditions for Nash equilibria, analogous to the Karush-Kuhn-Tucker (KKT) conditions in optimization. By relying on an augmented Lagrangian approach to handle constraints, the solver is able to solve multi-player games with several agents and a high level of interactions at real-time speeds. Finding a Nash equilibrium for 3 autonomous cars in a freeway merging scenario takes 90 ms. Our primary contributions are:

1) A general solver for dynamic games aimed at identifying Generalized Nash Equilibrium strategies.
2) A real time MPC implementation of the solver mitigating the "frozen robot" problem that arises in complex driving scenarios for non-game-theoretic MPC approach. (Fig. 1).
3) A comparison with iLQGames [4]. ALGAMES finds Nash equilibria 3 times faster than iLQGames for a fixed constraint satisfaction criterion.

II. Related Work

A. Equilibrium Selection

Recent work focused on solving multi-player dynamic games can be categorized by the type of equilibrium they select. Several works [2, 3, 9, 13] have opted to search for Stackelberg equilibria, which model an asymmetry of information between players. These approaches are usually formulated for games with two players, a leader and a follower. The leader chooses its strategy first, then the follower selects the best response to the leader's strategy. Alternatively, a Nash equilibrium does not introduce hierarchy between players; each player's strategy is the best response to the other players' strategies. As pointed out in [6], searching for open-loop Stackelberg equilibrium strategies can fail on simple examples. In the context of autonomous driving, for instance, when players' cost functions only depend on their own state and control trajectories, the solution becomes trivial. The leader ignores mutual collision constraints and the follower has to adapt to this strategy. This behavior can be overly aggressive for the leader (or overly passive for the follower) and does not capture the game-theoretic nature of the problem (see Figure 2).

Nash equilibria have been investigated in [4, 5, 8, 14, 15, 16, 17]. We also take the approach of searching for Nash equilibria, as this type of equilibrium seems better suited to symmetric, multi-robot interaction scenarios. Indeed, we have observed more natural behavior emerging from Nash equilibria compared to Stackelberg when solving for open-loop strategies.

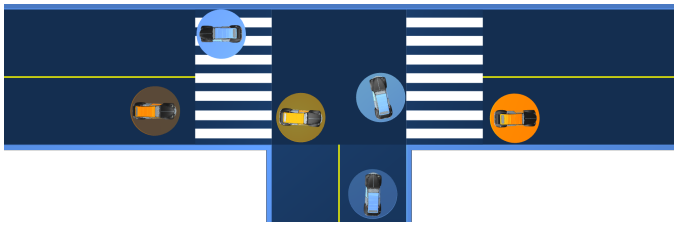

Fig. 2. Superimposed sequence of images depicting the trajectories obtained by solving for open-loop Stackelberg equilibrium strategies. The slow blue vehicle is the leader and cuts in front of the the fast orange vehicle, which is the follower. This example illustrates the fact that the leader always has the "right of way" over the follower in any situation. The Nash equilibrium strategy for this scenario lets the orange car cross the intersection first.

B. Game-Theoretic Trajectory Optimization

Most of the algorithms proposed in the robotics literature to solve for game-theoretic equilibria can be grouped into four types: First are algorithms aimed at finding Nash equilibria that rely on decomposition, such as Jacobi or Gauss-Siedel methods [8, 14, 18]. These algorithms are based on an iterative best response scheme in which players take turns at improving their strategies considering the other agents' strategies as immutable. This type of approach is easy to interpret and scales reasonably well with the number of players. However, convergence of these algorithms is not well understood [12], and special care is required to capture the game-theoretic nature of the problem [8]. Moreover, solving for a Nash equilibrium until convergence can require many iterations, each of which is a (possibly expensive) trajectory optimization problem. This can lead to prohibitively long solution times.

Second, there are a variety of algorithms based on dynamic programming. In [6], a Markovian Stackelberg strategy is computed via dynamic programming. This approach seems to capture the game-theoretic nature of autonomous driving. However, dynamic programming suffers from the curse of dimensionality and, therefore, practical implementations rely on simplified dynamics models coupled with coarse discretization of the state and input spaces. To counterbalance these approximations, a lower-level planner informed by the state values under the Markovian Stackelberg strategy is run. This approach, which scales exponentially with the state dimension, has only been demonstrated in a two-player setting. Adding more players is likely to prevent real-time application of this algorithm. In contrast, our proposed approach scales polynomially with the number of players (see Section IV-E).

Third, algorithms akin to differential dynamic programming have been developed for robust control [19] and later applied to game-theoretic problems [4, 15]. This approach scales polynomially with the number of players and is fast enough to run real-time in a model-predictive control (MPC) fashion [4]. However, contrary to ALGAMES, this type of approach does not natively handle constraints. Collision-avoidance con-

straints are typically handled using large penalties that can result in numerical ill-conditioning which, in turn, can impact the robustness or the convergence rate of the solver. Moreover, it leads to a trade-off between trajectory efficiency and avoiding collisions with other players.

Finally, algorithms that are analogous to direct methods in trajectory optimization have also been developed [16, 17]. An algorithm based on a first-order splitting method was proposed by Di [17] that is known to have a linear convergence rate. Di's experiments show convergence of the algorithm after typically 10^3 to 10^4 iterations. A different approach based on Newton's method has been proposed [16], but it is restricted to unconstrained dynamic games. Our solver belongs to this family of approaches. It also relies on a second-order Newton-type method, but it is able to handle general state and control input constraints. In addition, we demonstrate convergence on relatively complex problems in typically less than 10^2 iterations.

C. Generalized Nash Equilibrium Problems

We focus on finding Nash equilibria for multi-player games in which players are coupled through shared state constraints (such as collision-avoidance). Therefore, these problems are instances of Generalized Nash Equilibrium Problems (GNEPs). The operations research field has a rich literature on GNEPs [20, 21, 22, 23, 24]. Exact penalty methods have been proposed to solve GNEPs [21, 22]. Complex constraints such as those that couple players' strategies are handled using penalties, allowing solution of multi-player games jointly for all the players. However, these exact penalty methods require minimization of nonsmooth objective functions, which leads to slow convergence rates in practice.

In the same vein, a penalty approach relying on an augmented Lagrangian formulation of the problem has been advanced by Pang et al. [20]. This work, however, converts the augmented Lagrangian formulation to a set of KKT conditions, including complementarity constraints. The resulting constraint-satisfaction problem is solved with an off-the-shelf linear complementarity problem (LCP) solver that exploits the linearity of a specific problem. Our solver, in contrast, is not tailored for a specific example and can solve general GNEPs. It draws inspiration from the augmented Lagrangian formulation, which does not introduce nonsmooth terms in the objective function, enabling fast convergence. Moreover, this formulation avoids ill-conditioning, which improves the numerical robustness of our solver.

III. Problem Statement

In the discretized trajectory optimization setting with N time steps, we denote by n the state size, m the control input size, x_k^ν the state, and u_k^ν the control input of player ν at the time step k. In formulating the game, we do not distinguish between the robot carrying out the computation, and the other agents whose trajectories it is predicting. All agents are modeled equivalently, as is typical in the case of Nash style games.

Following the formalism of Facchinei [12], we consider the GNEP with M players. Each player ν decides over its control input variables $U^\nu = [(u_1^\nu)^T \dots (u_{N-1}^\nu)^T]^T \in \mathbb{R}^{\bar{m}^\nu}$. This is player ν's strategy where m^ν denotes the dimension of the control inputs controlled by player ν and $\bar{m}^\nu = m^\nu(N-1)$ is the dimension of the whole trajectory of player ν's control inputs. By $U^{-\nu}$, we denote the vector of all the players' strategies except the one of player ν. Additionally, we define the trajectory of state variables $X = [(x_2)^T \dots (x_N)^T]^T \in \mathbb{R}^{\bar{n}}$ where $\bar{n} = n(N-1)$, which results from applying all the control inputs decided by the players to a joint dynamical system,

$$x_{k+1} = f(x_k, u_k^1, \dots, u_k^M) = f(x_k, u_k), \tag{1}$$

with k denoting the time-step index. The kinodynamic constraints over the whole trajectory can be expressed with $\bar{n}$ equality constraints,

$$D(X, U^1, \dots, U^M) = D(X, U) = 0 \in \mathbb{R}^{\bar{n}}. \tag{2}$$

The cost function of each player is noted $J^\nu(X, U^\nu) : \mathbb{R}^{\bar{n}+\bar{m}^\nu} \to \mathbb{R}$. It depends on player ν's control inputs U^ν as well as on the state trajectory X, which is shared with all the other players. The goal of player ν is to select a strategy U^ν and a state trajectory X that minimizes the cost function J^ν. Naturally, the choice of state trajectory X is constrained by the other players' strategies $U^{-\nu}$ and the dynamics of the system via Equation 2. In addition, the strategy U^ν must respect a set of constraints that depends on the state trajectory X as well as on the other players strategies $U^{-\nu}$ (e.g., collision-avoidance constraints). We express this with a concatenated set of inequality constraints $C : \mathbb{R}^{\bar{n}+\bar{m}} \to \mathbb{R}^{n_c}$. Formally,

$$\begin{aligned} \min_{X, U^\nu} \quad & J^\nu(X, U^\nu), \\ \text{s.t.} \quad & D(X, U) = 0, \\ & C(X, U) \leq 0. \end{aligned} \tag{3}$$

The set of M Problems (3), forms a GNEP because of the constraints that couple the strategies of all the players. A solution of this GNEP (a generalized Nash equilibrium), is a vector $\hat{U}$ such that, for all $\nu = 1, \dots, M$, $\hat{U}^\nu$ is a solution to (3) with the other players' strategies fixed to $\hat{U}^{-\nu}$. This means that at an equilibrium point $\hat{U}$, no player can decrease their cost by unilaterally changing their strategy U^ν to any other feasible point.

When solving for a generalized Nash equilibrium of the game, U, we identify open-loop Nash equilibrium trajectories, in the sense that the whole trajectory U^ν is the best response to the other players' strategies $U^{-\nu}$ given the initial state of the system x_0. Thus, the control signal is a function of time, not of the current state of the system[1] x_k. However, one can repeatedly resolve the open-loop game as new information is obtained over time to obtain a policy that is closed-loop in the model-predictive control sense, as demonstrated in Section

[1] One might also explore solving for feedback Nash equilibria, where the strategies are functions of the state of all agents. This is an interesting direction for future work.

VII. This formulation is general enough to comprise multi-player general-sum dynamic games with nonlinear constraints on the states and control inputs. Practically, in the context of autonomous driving, the cost function J^ν encodes the objective of player ν, while the concatenated set of constraints C includes collision constraints coupled between players. We assume differentiability of the constraints and twice differentiability of the cost functions.

IV. Augmented Lagrangian Formulation

We propose an algorithm to solve the previously defined GNEP in the context of trajectory optimization. We express the condition that players are acting optimally to minimize their cost functions subject to constraints as an equality. To do so, we first derive the augmented Lagrangian associated with (3) solved by each player. Then, we use the fact that, at an optimal point, the gradient of the augmented Lagrangian is null [25]. Therefore, at a generalized Nash equilibrium point, the gradients of the augmented Lagrangians of all players must be null. Concatenating this set of M equality constraints with the dynamics equality constraints, we obtain a set of equations that we solve using a quasi-Newton root-finding algorithm.

A. Individual Optimality

First, without loss of generality, we suppose that the vector C is actually the concatenated set of inequality and equality constraints, i.e., $C = [C_i^T\, C_e^T]^T \in \mathbb{R}^{n_{ci}+n_{ce}}$, where $C_i \leq 0$ is the vector of inequality constraints and $C_e = 0$ is the vector of equality constraints. To embed the notion that each player is acting optimally, we formulate the augmented Lagrangian associated with (3) for player ν. The dynamics constraints are handled with the Lagrange multiplier term $\mu^\nu \in \mathbb{R}^{\bar{n}}$, while the other constraints are dealt with using both a multiplier and a quadratic penalty term specific to the augmented Lagrangian formulation. As a motivation for this differential treatment; one typically handles inequality and highly nonlinear equality constraints with an augmented Lagrangian formulation for its improved robustness. We denote by $\lambda \in \mathbb{R}^{n_c}$ the Lagrange multipliers associated with the vector of constraints C; $\rho \in \mathbb{R}^{n_c}$ is a penalty weight;

$$L^\nu(X,U) = J^\nu + \mu^{\nu T} D + \lambda^T C + \frac{1}{2} C^T I_\rho C. \tag{4}$$

I_ρ is a diagonal matrix defined as,

$$I_{\rho,kk} = \begin{cases} 0 & \text{if } C_k(X,U) < 0 \wedge \lambda_k = 0,\, k \leq n_{ci}, \\ \rho_k & \text{otherwise,} \end{cases} \tag{5}$$

where $k = 1,\ldots,n_{ci}+n_{ce}$ indicates the kth constraint. It is important to notice that the Lagrange multipliers μ^ν associated with the dynamics constraints are specific to each player ν, but the Lagrange multipliers and penalties λ and ρ are common to all players. Given the appropriate Lagrange multipliers μ^ν and λ, the gradient of the augmented Lagrangian with respect to the individual decision variables $\nabla_{X,U^\nu} L^\nu = G^\nu$ is null at an optimal point of (3). The fact that player ν is acting optimally to minimize J^ν under the constraints D and C can therefore be expressed as follows,

$$\nabla_{X,U^\nu} L^\nu(X,U,\mu^\nu) = G^\nu(X,U,\mu^\nu) = 0. \tag{6}$$

It is important to note that this equality constraint preserves coupling between players since the gradient G^ν depends on the other players' strategies $U^{-\nu}$.

B. Root-Finding Problem

At a generalized Nash equilibrium, all players are acting optimally and the dynamics constraints are respected. Therefore, to find an equilibrium point, we have to solve the following root-finding problem,

$$\begin{aligned} \min_{X,U,\mu} \quad & 0, \\ \text{s.t.} \quad & G^\nu(X,U,\mu^\nu) = 0,\ \forall\, \nu \in \{1,\ldots,M\}, \\ & D(X,U) = 0, \end{aligned} \tag{7}$$

We use Newton's method to solve the root-finding problem. We denote by G the concatenation of the augmented Lagrangian gradients of all players and the dynamics constraints, $G(X,U,\mu) = [(G^1)^T,\ldots,(G^M)^T, D^T]^T$, where $\mu = [(\mu^1)^T,\ldots,(\mu^M)^T]^T \in \mathbb{R}^{\bar{n}M}$. We compute the first-order derivative of G with respect to the primal variables X,U and the dual variables μ that we concatenate in a single vector $y = [(X)^T,(U)^T,(\mu)^T]$,

$$H = \nabla_{X,U,\mu} G = \nabla_y G. \tag{8}$$

Newton's method allows us to identify a search direction δy in the primal-dual space,

$$\delta y = -H^{-1} G. \tag{9}$$

We couple this search direction with a backtracking line-search [26] given in Algorithm 1 to ensure local convergence to a solution using Newton's Method [26] detailed in Algorithm 2.

Algorithm 1 Backtracking line-search

```
procedure LineSearch(y, G, δy)
    Parameters
    α = 1,
    β ∈ (0, 1/2),
    τ ∈ (0, 1),
    Until ||G(y + αδy)||_1 < (1 − αβ)||G(y)||_1 do
        α ← τα
    return α
```

Algorithm 2 Newton's method for root-finding problem

```
procedure Newton'sMethod(y)
    Until Convergence do
        G ← [(∇_{X,U^1} L^1)^T, ..., (∇_{X,U^M} L^M)^T, D^T]^T
        H ← ∇_y G
        δy ← −H^{−1} G
        α ← LineSearch(y, G, δy)
        y ← y + αδy
    return y
```

Algorithm 3 ALGAMES solver

procedure ALGAMES(y_0, ρ_0)
  Initialization
  $\rho \leftarrow \rho^{(0)}$,
  $\lambda \leftarrow 0$,
  $\mu^\nu \leftarrow 0$, $\quad \forall \nu$
  $X, U \leftarrow X^{(0)}, U^{(0)}$
  Until Convergence **do**
    $y \leftarrow$ NEWTON'SMETHOD(y)
    $\lambda \leftarrow$ DUALASCENT(y, λ, ρ), $\quad \triangleright$ Equation 10
    $\rho \leftarrow$ INCREASINGSCHEDULE(ρ), $\triangleright$ Equation 11
  return y

C. Augmented Lagrangian Updates

To obtain convergence of the Lagrange multipliers λ, we update them with a dual-ascent step. This update can be seen as shifting the value of the penalty terms into the Lagrange multiplier terms,

$$\lambda_k \leftarrow \begin{cases} \max(0, \lambda_k + \rho_k C_k(X,U)) & k \leq n_{ci}, \\ \lambda_k + \rho_k C_k(X,U) & n_{ci} < k \leq n_{ci} + n_{ce}. \end{cases} \tag{10}$$

We also update the penalty weights according to an increasing schedule, with $\gamma > 1$:

$$\rho_k \leftarrow \gamma \rho_k, \ \forall k \in \{1, \dots, n_c\}. \tag{11}$$

D. ALGAMES

By combining Newton's method for finding the point where the dynamics is respected and the gradients of the augmented Lagrangians are null with the Lagrange multiplier and penalty updates, we obtain our solver ALGAMES (Augmented Lagrangian GAME-theoretic Solver) presented in Algorithm 3. The algorithm, which iteratively solves the GNEP, requires as inputs an initial guess for the primal-dual variables $y^{(0)}$ and initial penalty weights $\rho^{(0)}$. The algorithm outputs the open-loop strategies of all players X, U and the Lagrange multipliers associated with the dynamics constraints μ.

E. Algorithm Complexity

Following a quasi-Newton approximation of the matrix H [26], we neglect some of the second-order derivative terms associated with the constraints. Therefore, the most expensive part of the algorithm is the Newton step defined by Equation 9. By exploiting the sparsity pattern of the matrix H, we can solve Equation 9 in $O(N(n+m)^3)$. Indeed, the sparsity structure allows us to perform a back-substitution scheme akin to solving a Riccati equation, which has known complexity of $O(N(n+m)^3)$. The complexity is cubic in the number of states n and the number of control inputs m, which are typically linear in the number of players M. Therefore, the overall complexity of the algorithm is $O(NM^3)$.

F. Algorithm Discussion

Here we discuss the inherent difficulty in solving for Nash equilibria in large problems, and explain some of the limitations of our approach. First of all, finding a Nash equilibrium is a non-convex problem in general. Indeed, it is known that even for single-shot discrete games, solving for exact Nash equilibria is computationally intractable for a large number of players [27]. It is therefore not surprising that, in our more difficult setting of a dynamic game in continuous space, no guarantees can be provided about finding an exact Nash equilibrium. Furthermore, in complex interaction spaces, constraints can be highly nonlinear and nonconvex. This is the case in the autonomous driving context with collision avoidance constraints. In this setting, one cannot expect to find an algorithm working in polynomial time with guaranteed convergence to a Nash equilibrium respecting constraints. On the other hand, *local* convergence of Newton's method to open-loop Nash equilibria has been established in the unconstrained case (that is, starting sufficiently close to the equilibrium, the algorithm will converge to it) [16]. Our approach solves a sequence of unconstrained problems via the augmented Lagrangian formulation. Each of these problems, therefore, has guaranteed *local* convergence. However, as expected, the overall method has no guarantee of global convergence to a generalized Nash equilibrium.

Second, our algorithm requires an initial guess for the state and control input trajectories X, U and the dynamics multipliers μ. Empirically, we observe that choosing $\mu = 0$ and simply rolling out the dynamics starting from the initial state x_0 without any control was a sufficiently good initial guess to get convergence to a local optimum that respects both the constraints and the first-order optimality conditions. For a detailed empirical study of the convergence of ALGAMES and its failure cases, we refer to Sections VI-D and VI-E.

Finally, even for simple linear quadratic games, the Nash equilibrium solution is not necessarily unique. In general, an entire subspace of equilibria exists. In this case, the matrix H in Equation 9 will be singular. In practice, we regularize this matrix so that large steps δy are penalized, resulting in an invertible matrix H and convergence to a Nash equilibrium that minimizes the norm of y.

V. Simulations: Design and Setup

We choose to apply our algorithm in the autonomous driving context. Indeed, many maneuvers like lane changing, ramp merging, overtaking, and intersection crossing involve a high level of interaction between vehicles. We assume a single car is computing the trajectories for all cars in its neighborhood, so as to find its own trajectory to act safely among the group. We assume that this car has access to a relatively good estimate of the surrounding cars' objective functions. Such an estimate could, in principle, be obtained by applying inverse optimal control on observed trajectories of the surrounding cars.

In a real application, the car would compute its strategy as frequently as possible in a receding-horizon loop to adapt to unforeseen changes in the environment. We demonstrate

the feasibility of this approach on complex driving scenarios where a classical predict-then-plan architecture fails to overcome the "frozen robot" problem.

A. Autonomous Driving Problem

1) Constraints: Each vehicle in the scene is an agent of the game. Our objective is to find a generalized Nash equilibrium trajectory for all of the vehicles. These trajectories have to be dynamically feasible. The dynamics constraints at time step k are expressed as follows,

$$x_{k+1} = f(x_k, u_k^1, \ldots, u_k^M). \tag{12}$$

We consider a nonlinear unicycle model for the dynamics of each vehicle. A vehicle state, x_k^ν, is composed of a 2D position, a heading angle and a scalar velocity. The control input u_k^ν is composed of an angular velocity and a scalar acceleration.

In addition, it is critical that the trajectories respect collision-avoidance constraints. We model the collision zone of the vehicles as circles of radius r. The collision constraints between vehicles are then simply expressed in terms of the position $\tilde{x}_k^\nu$ of each vehicle,

$$r^2 - ||\tilde{x}_k^\nu - \tilde{x}_k^{\nu'}||_2^2 \leq 0, \quad \forall\, \nu, \nu' \in \{1, \ldots, M\}, \nu \neq \nu'. \tag{13}$$

We also model boundaries of the road to force the vehicles to remain on the roadway. This means that the distance between the vehicle and the closest point, q, on each boundary, b, has to remain larger than the collision circle radius, r,

$$r^2 - ||\tilde{x}_k^\nu - q_b||_2^2 \leq 0, \quad \forall\, b, \forall\, \nu \in \{1, \ldots, M\}. \tag{14}$$

In summary, based on reasonable simplifying assumptions, we have expressed the driving problem in terms of non-convex and non-linear coupled constraints.

2) Cost Function: We use a quadratic cost function penalizing the use of control inputs and the distance between the current state and the desired final state x_f of the trajectory,

$$J^\nu(X, U^\nu) = \sum_{k=1}^{N-1} \frac{1}{2}(x_k - x_f)^T Q (x_k - x_f) + \frac{1}{2} u_k^{\nu T} R u_k^\nu + \tag{15}$$

$$\frac{1}{2}(x_N - x_f)^T Q_f (x_N - x_f). \tag{16}$$

This cost function only depends on the decision variables p^ν of vehicle ν. Players' behaviors are coupled only through collision constraints. We could also add terms depending on other vehicles' strategies, such as a congestion penalty.

VI. Comparison to iLQGames

A. Motivation

In order to evaluate the merits of ALGAMES, we compare it to iLQGames [4] which is a DDP-based algorithm for solving general dynamic games. Both algorithms solve the problem by iteratively solving linear-quadratic approximations that have an analytical solution [11]. For iLQGames, the augmented objective function $\hat{J}^\nu$ differs from the objective function, J^ν, by a quadratic term penalizing constraint violations,

$$\hat{J}^\nu(X, U) = J^\nu(X, U) + \frac{1}{2} C(X, U)^T I_\rho C(X, U). \tag{17}$$

Where I_ρ is defined by,

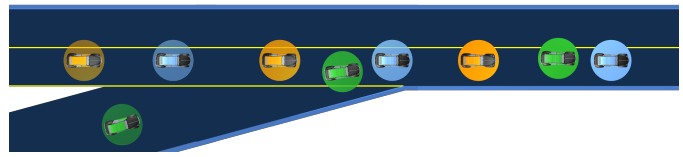

Fig. 3. On the left, the three cars at their nominal initial state. On the right, the three cars are standing at the desired final state. The green car has successfully merged in between the two other cars. The roadway boundaries are depicted in light blue.

Scenario	# Players	ALGAMES	iLQGames
Ramp merging	2	$\mathbf{38 \pm 10}$**ms**	$104 \pm 23ms$
	3	$\mathbf{89 \pm 14}$**ms**	$197 \pm 15ms$
	4	$860 \pm 251ms$	$\mathbf{705 \pm 209}$**ms**
Intersection	2	$\mathbf{50 \pm 11}$**ms**	$752 \pm 168ms$
	3	$\mathbf{116 \pm 22}$**ms**	$362 \pm 93ms$
	4	$\mathbf{509 \pm 33}$**ms**	$1905 \pm 498ms$

Fig. 4. For each scenario and each number of players, we run each solver 100 times to estimate the mean solve time and its standard deviation.

$$I_{\rho,kk} = \begin{cases} 0 & \text{if } C_k(X, U) < 0, k \leq n_{ci}, \\ \rho_k & \text{otherwise.} \end{cases} \tag{18}$$

Here ρ is an optimization hyperparameter that we can tune to satisfy constraints. For ALGAMES, the augmented objective function, L^ν, is actually an augmented Lagrangian, see Equation 4. The hyperparameters for ALGAMES are the initial value of $\rho^{(0)}$ and its increase rate γ defined in Equation 11.

B. Timing Experiments

We evaluate the performance of both algorithms in two scenarios, see Figures 3 and 7, with the number of players varying from two to four. To compare the speed of both algorithms, we set the termination criterion as a threshold on constraint violations $C \leq 10^{-3}$. The timing results averaged over 100 samples are presented in Table 4. First, we notice that both algorithms achieve real-time or near-real-time performance on complex autonomous driving scenarios (the horizon of the solvers is fixed to $5s$).

We observe that the speed performance of ALGAMES and iLQGames are comparable in the ramp merging scenario. For this scenario, we tuned the value of the penalty for iLQGames to $\rho = 10^2$. Notice that for all scenarios the dimensions of the problem are scaled so that the velocities and displacements are all the same order of magnitude. For the intersection scenario, we observe that the two-player and four-player cases both have much higher solve times for iLQGames compared to the 3-player case. Indeed, in those two cases, we had to increase the penalty to $\rho = 10^3$, otherwise the iLQGames would plateau and never reach the constraint satisfaction criterion. This, in turn, slowed the algorithm down by decreasing the constraint violation convergence rate.

C. Discussion

The main takeaway from these experiments is that, for a given scenario, it is generally possible to find a suitable value for ρ that will ensure the convergence of iLQGames to constraint satisfaction. With higher values for ρ, we can reach

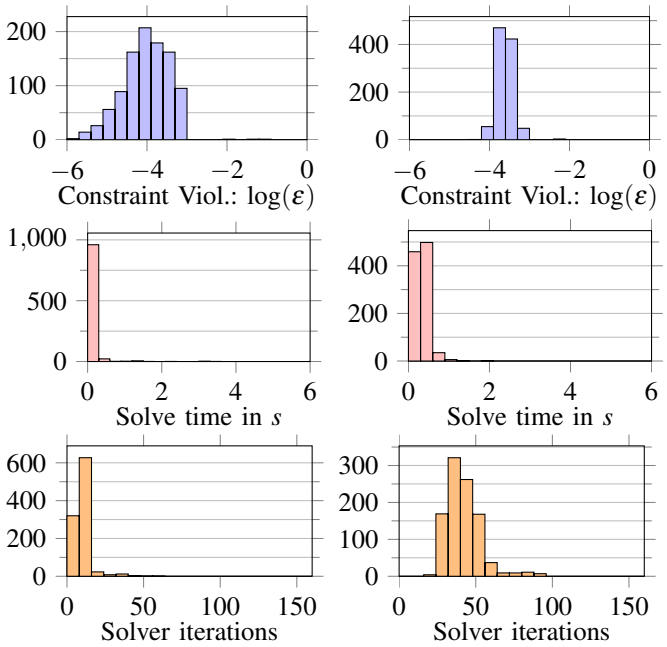

Fig. 5. Monte Carlo analysis with 1000 randomly sampled initial states of ALGAMES on the left and iLQGames on the right. The top plots indicate the largest constraint violation of the solution at the end of the solve, $\varepsilon \geq 0$ (smaller ε means constraints are better satisfied). Middle plots show the solve time. The bottom left and right plots displays the number of Newton steps and the number of Riccati backward passes executed during the solve of ALGAMES and iLQGames respectively.

better constraint satisfaction at the expense of slower convergence rate. In the context of a receding horizon implementation (MPC), finding a good choice of ρ that would suit the whole sequence of scenarios encountered by a vehicle could be difficult. In contrast, the same hyperparameters $\rho^{(0)} = 1$ and $\gamma = 10$ were used in ALGAMES for all the experiments across this paper. This supports the idea that, thanks to its adaptive penalty scheme, ALGAMES requires little tuning.

While performing the timing experiments, we also noticed several instances of oscillatory behavior for iLQGames. The solution would oscillate, preventing it from converging. This happened even after an adaptive regularization scheme was implemented to regularize iLQGames' Riccati backward passes. Oscillatory behavior was not seen with ALGAMES. We hypothesize that this is due to the dual ascent update coupled with the penalty logic detailed in Equations 10 and 5, which add hysteresis to the solver.

D. Monte Carlo Analysis

To evaluate the robustness of ALGAMES, we performed a Monte Carlo analysis of its performance on a ramp merging problem. First, we set up a roadway with hard boundaries as pictured in Fig. 3. We position two vehicles on the roadway and one on the ramp in a collision-free initial configuration. We choose a desired final state where the incoming vehicle has merged into the traffic. Our objective is to generate generalized Nash equilibrium trajectories for the three vehicles. These trajectories are collision-free and cannot be improved unilaterally by any player. To introduce randomness in the solving process, we apply a random perturbation to the initial state of the problem. Specifically, we perturb x_0 by adding a uniformly sampled noise. This would typically correspond to displacing the initial position of the vehicles by $\pm 1m$, changing their initial velocity by $\pm 3\%$ and their heading by $\pm 2.5°$.

Scenario	Freq. in Hz	$\mathbb{E}[\delta t]$ in ms	$\sigma[\delta t]$ in ms
Ramp Merging	69	14	72
Intersection	66	15	66

Fig. 6. Running the MPC implementation of ALGAMES 100 times on both scenarios, we obtain the mean update frequency of the MPC as well as the mean and standard deviation of δt, the time required to update the MPC plan.

We observe in Figure 5, that ALGAMES consistently finds a satisfactory solution to the problem using the same hyperparameters $\rho^{(0)} = 1$ and $\gamma = 10$. Out of the 1000 samples 99.5% converged to constraint satisfaction $C \leq 10^{-3}$ while respecting the optimality criterion $||G||_1 < 10^{-2}$. By definition, $||G||_1$ is a merit function for satisfying optimality and dynamics constraints. We also observe that the solver converges to a solution in less than $0.2s$ for 96% of the samples. The solver requires less than 16 Newton steps to converge for 94% of the samples. These empirical data tend to support the fact that ALGAMES is able to solve the class of ramp merging problem quickly and reliably.

For comparison, we present in Figure 5 the results obtained with iLQGames. We apply the same constraint satisfaction criterion $C \leq 10^{-3}$. We fixed the value of the penalty hyperparameter ρ for all the samples as it would not be a fair comparison to tune it for each sample. Only 3 samples did not converge with iLQGames, this is a performance comparable to ALGAMES for which 5 samples failed to converge. However, we observe that iLQGames is 3 times slower than ALGAMES with an average solve time of 350 ms compared to 110 ms and require on average 4 times more iterations (9 against 41).

E. Solver Failure Cases

The Monte Carlo analysis allows us to identify the typical failure cases of our solver. We empirically identify the cases where the solver does not satisfy the constraints or the optimality criterion for the ramp merging problem. Typically in such cases, the initial guess, which consists of rolling out the dynamics with no control, is far from a reasonable solution. Since the constraints are ignored during this initial rollout, the car at the back can overtake the car at the front by driving through it. This creates an initial guess where constraints are strongly violated. Moreover, we hypothesize that the tight roadway boundary constraints tend to strongly penalize solutions that would 'disentangle' the car trajectories because they would require large boundary violation at first. Therefore, the solver gets stuck in this local optimum where cars overlap each other. Sampling several initial guesses with random initial control inputs and solving in parallel could reduce the occurrence of these failure cases. Also, being able to detect, reject, and re-sample initial guesses when the initial car trajectories are strongly entangled could also improve the robustness of the solver.

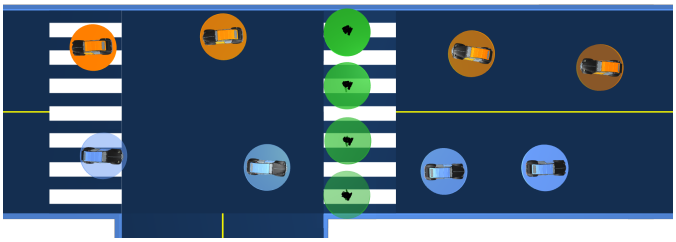

Fig. 7. The blue car starts on the left and finishes on the right. The orange does the opposite. The pedestrian with the green collision avoidance cylinder crosses the road from the bottom to the top of the image.

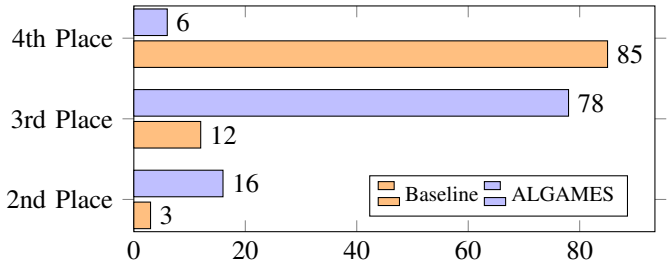

Fig. 8. Monte Carlo analysis with 100 randomly sampled initial states, we record the position of the merging vehicle in the traffic at the end of the 6-second simulations.

VII. MPC Implementation of ALGAMES

In this section, we propose a model-predictive control (MPC) implementation of the algorithm which provides us with a feedback policy instead of an open-loop strategy and demonstrates real-time performance. We compare this MPC to a non-game-theoretic baseline on a crowded ramp merging which is known to be conducive to the "frozen robot" problem.

A. MPC Feedback Policy

The strategies identified by ALGAMES are open-loop Nash-equilibrium strategies. They are sequences of control inputs. On the contrary, DDP-based approaches like iLQGames, solve for feedback Nash-equilibrium strategies that provide a sequence of control gains. In the MPC setting, we can obtain a feedback policy with ALGAMES by updating the strategy as fast as possible and only executing the beginning of the strategy. This assumes a fast update rate of the solution. To support the feasibility of the approach, we implemented an MPC on the ramp merging scenario described in Figure 3. There are 3 players constantly maintaining a 40 time step strategy with 3 seconds of horizon. We simulate 3 seconds of operation of the MPC by constantly updating the strategies and propagating noisy unicycle dynamics for each vehicle. We compile the results from 100 MPC trajectories in Table 6. We obtain a 69 Hz update frequency for the planner on average. We observe similar performance on the intersection problem defined in Figure 7, with an update frequency of 66 Hz.

B. "Unfreezing" the Robot

To illustrate the benefits of using ALGAMES in a receding-horizon loop, we compare it to a non-game-theoretic baseline MPC. With this baseline, the prediction step and the planning step are decoupled. Specifically, each agent predicts the trajectories of the surrounding vehicles by propagating straight, constant velocity trajectories. Then, each agent plans for itself assuming these predicted trajectories are immutable obstacles. We test these two controllers on a challenging scenario where a vehicle has to merge on a crowded highway as presented in Figure 1. We perform a Monte Carlo analysis by uniformly sampling the initial state, x_0, around a nominal state with perturbations corresponding to a $\pm 2.5m$ longitudinal displacement, $\pm 25cm$ lateral displacement, $\pm 3^\circ$ in angular displacement for each car. Given the initial state, the vehicle on the ramp should be able to merge between the blue and orange cars or the orange and green cars, taking the 2nd and 3rd place respectively. However, waiting for all cars to pass before merging into 4th place is not a desirable behavior. Indeed, with such a policy, the merging vehicle has to slow down significantly and could get stuck on the ramp if the highway does not clear. We run ALGAMES in a receding horizon loop and the baseline MPC to generate 6-second trajectories for 100 different initial states. We record the position of the merging vehicle at the end of the simulation and compile the results in Figure 8.

We observe that the "frozen robot" problem occurs with the baseline MPC for 85% of the simulations. An interpretation of this result is that the vehicle on the ramp cannot find a merging maneuver that is not colliding with its constant-velocity trajectory predictions. Since there is no feasible merging maneuver, the only option left is to wait for the other vehicles to pass before merging.

On the contrary, by running ALGAMES in a receding-horizon loop, the merging vehicle inserts into traffic in 2nd or 3rd place in 96% of the simulations (Figure 8). ALGAMES avoids the "frozen robot" pitfall in most cases by gradually adjusting its velocity to merge with minimal disruption to the traffic (Figure 1).

VIII. Conclusions

We have introduced a new algorithm for finding constrained Nash-equilibrium trajectories in multi-player dynamic games. We demonstrated the performance and robustness of the solver through a Monte Carlo analysis on complex autonomous driving scenarios including nonlinear and non-convex constraints. We have shown real-time performance for up to 4 players and implemented ALGAMES in a receding-horizon framework to give a feedback policy. We empirically demonstrated the ability of ALGAMES to mitigate the "frozen robot" problem in comparison to a non-game-theoretic receding horizon planner. The results we obtained from ALGAMES are promising, as they seem to let the vehicles share the responsibility for avoiding collisions, leading to natural-looking trajectories where players are able to negotiate complex, interactive traffic scenarios that are challenging for traditional, non-game-theoretic trajectory planners. For this reason, we believe that ALGAMES could be a very efficient tool to generate trajectories in situations where the level of interaction between players is strong. Our implementation of ALGAMES is available at `https://github.com/RoboticExplorationLab/ALGAMES.jl`.

References

[1] P. Trautman and A. Krause, "Unfreezing the robot: Navigation in dense, interacting crowds," in *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, (Taipei), pp. 797–803, IEEE, Oct. 2010.

[2] D. Sadigh, S. S. Sastry, S. A. Seshia, and A. Dragan, "Information gathering actions over human internal state," in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, (Daejeon, South Korea), pp. 66–73, IEEE, Oct. 2016.

[3] D. Sadigh, S. Sastry, S. A. Seshia, and A. D. Dragan, "Planning for Autonomous Cars that Leverage Effects on Human Actions," in *Robotics: Science and Systems XII*, Robotics: Science and Systems Foundation, 2016.

[4] D. Fridovich-Keil, E. Ratner, J. C. Shih, A. D. Dragan, and C. J. Tomlin, "Iterative Linear Quadratic Approximations for Nonlinear Multi-Player General-Sum Differential Games," *arXiv preprint arXiv:1909.04694*, p. 8, 2019.

[5] A. Dreves and M. Gerdts, "A generalized Nash equilibrium approach for optimal control problems of autonomous cars: A generalized Nash equilibrium approach for optimal control problems of autonomous cars," *Optimal Control Applications and Methods*, vol. 39, pp. 326–342, Jan. 2018.

[6] J. F. Fisac, E. Bronstein, E. Stefansson, D. Sadigh, S. S. Sastry, and A. D. Dragan, "Hierarchical Game-Theoretic Planning for Autonomous Vehicles," in *2019 International Conference on Robotics and Automation (ICRA)*, (Montreal, QC, Canada), pp. 9590–9596, IEEE, May 2019.

[7] E. Schmerling, K. Leung, W. Vollprecht, and M. Pavone, "Multimodal Probabilistic Model-Based Planning for Human-Robot Interaction," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, (Brisbane, QLD), pp. 1–9, IEEE, May 2018.

[8] R. Spica, D. Falanga, E. Cristofalo, E. Montijano, D. Scaramuzza, and M. Schwager, "A Real-Time Game Theoretic Planner for Autonomous Two-Player Drone Racing," *arXiv:1801.02302 [cs]*, Jan. 2018.

[9] A. Liniger and J. Lygeros, "A Noncooperative Game Approach to Autonomous Racing," *IEEE Transactions on Control Systems Technology*, pp. 1–14, 2019.

[10] A. Bressan, "Noncooperative Differential Games. A Tutorial," *Department of Mathematics, Penn State University*, p. 81, 2010.

[11] T. Basar and G. J. Olsder, *Dynamic noncooperative game theory*, vol. 23. Siam, 1999.

[12] F. Facchinei and C. Kanzow, "Generalized Nash equilibrium problems," *4OR*, vol. 5, pp. 173–210, Sept. 2007.

[13] J. H. Yoo and R. Langari, "Stackelberg Game Based Model of Highway Driving," in *Volume 1: Adaptive Control; Advanced Vehicle Propulsion Systems; Aerospace Systems; Autonomous Systems; Battery Modeling; Biochemical Systems; Control Over Networks; Control Systems Design; Cooperativ*, (Fort Lauderdale, Florida, USA), pp. 499–508, ASME, Oct. 2012.

[14] A. Britzelmeier, A. Dreves, and M. Gerdts, "Numerical solution of potential games arising in the control of cooperative automatic vehicles," in *2019 Proceedings of the Conference on Control and Its Applications* (W. S. Levine and R. Stockbridge, eds.), (Philadelphia, PA), pp. 38–45, Society for Industrial and Applied Mathematics, Jan. 2019.

[15] B. Di and A. Lamperski, "Differential Dynamic Programming for Nonlinear Dynamic Games," *arXiv:1809.08302 [math]*, Sept. 2018.

[16] B. Di and A. Lamperski, "Newton's Method and Differential Dynamic Programming for Unconstrained Nonlinear Dynamic Games," *arXiv:1906.09097 [cs, eess]*, Jan. 2020.

[17] B. Di and A. Lamperski, "First-Order Algorithms for Constrained Nonlinear Dynamic Games," *arXiv:2001.01826 [cs, eess]*, Jan. 2020.

[18] M. Wang, Z. Wang, J. Talbot, J. Christian Gerdes, and M. Schwager, "Game Theoretic Planning for Self-Driving Cars in Competitive Scenarios," in *Robotics: Science and Systems XV*, Robotics: Science and Systems Foundation, June 2019.

[19] J. Morimoto and C. G. Atkeson, "Minimax Differential Dynamic Programming: An Application to Robust Biped Walking," *Advances in neural information processing systems*, pp. 1563–1570, 2003.

[20] J.-S. Pang and M. Fukushima, "Quasi-variational inequalities, generalized Nash equilibria, and multi-leader-follower games," *Computational Management Science*, vol. 2, pp. 21–56, Jan. 2005.

[21] F. Facchinei and J.-S. Pang, "Exact penalty functions for generalized Nash problems," *Large-scale nonlinear optimization*, pp. 115–126, 2006.

[22] F. Facchinei, A. Fischer, and V. Piccialli, "Generalized Nash equilibrium problems and Newton methods," *Mathematical Programming*, vol. 117, pp. 163–194, Mar. 2009.

[23] F. Facchinei and C. Kanzow, "Penalty Methods for the Solution of Generalized Nash Equilibrium Problems," *SIAM Journal on Optimization*, vol. 20, pp. 2228–2253, Jan. 2010.

[24] M. Fukushima, "Restricted generalized Nash equilibria and controlled penalty algorithm," *Computational Management Science*, vol. 8, pp. 201–218, Aug. 2011.

[25] D. P. Bertsekas, *Constrained optimization and Lagrange multiplier methods*. Academic press, 2014.

[26] J. Nocedal and S. Wright, *Numerical optimization*. Springer Science & Business Media, 2006.

[27] C. Daskalakis, P. W. Goldberg, and C. H. Papadimitriou, "The complexity of computing a Nash equilibrium," *SIAM Journal on Computing*, vol. 39, no. 1, pp. 195–259, 2009.