Contents lists available at ScienceDirect

Nonlinear Analysis: Hybrid Systems

journal homepage: www.elsevier.com/locate/nahs

Graph-based distributed Nash equilibrium seeking for potential games☆,☆☆

T. Sântejudean [a,b,*], V.S. Varma [b,a], I.C. Morărescu [b,a], L. Buşoniu [a,c,*]

[a] Technical University of Cluj–Napoca, Romania
[b] Université de Lorraine, CNRS, CRAN, F-54000 Nancy, France
[c] Corresponding Member of the Romanian Academy, Bucharest, Romania

ARTICLE INFO

Keywords:
Nash equilibrium seeking
Distributed optimization
Networked systems

ABSTRACT

We introduce Distributed Asynchronous Potential function Decrease (DAPD), a discrete-time, graph-based algorithm for finding pure Nash equilibria in ordinal potential games. Two different settings are studied: one in which cost functions are twice-differentiable and convex, with Lipschitz first derivatives, and another when costs are only Lipschitz-continuous and may be non-convex. A novel graph-based update scheduler is proposed, which accelerates DAPD convergence by allowing parallel, decentralized updates of non-neighboring players. The scheduler chooses the next player to update in each neighborhood as the one with the largest decrease in cost function at the previous update. The graph topology is fixed, connected and undirected, and the update of each player depends only on its neighbors. We prove that when run with the proposed scheduler, DAPD converges to a pure Nash equilibrium in the differentiable setting, and to an ϵ-Nash equilibrium in the Lipschitz setting. In numerical experiments, DAPD with the new scheduler converges faster than with a round-robin scheduler, while better balancing the number of computations and communications than several synchronous and asynchronous baselines.

1. Introduction

We consider a multi-agent, distributed optimization problem defined as an ordinal potential game. Distributed optimization methods reach a pure Nash Equilibrium (NE) under appropriate assumptions [1,2]. Such methods were extensively studied by [3–5], where multiple agents aim to optimize a global objective, usually taken as the sum of all agents' objective (cost) functions, by using consensus-type dynamics [5,6]. These algorithms have been used to solve various optimization problems, such as large-scale matrix factorization [7] and linear programs [8]. Recently, distributed synchronous methods and their convergence to differential NE were studied by [9,10]. Although computational efficiency is better due to all agents computing the updates in parallel, these methods are often communication-intensive [9]. The number of communications required scales with the number of agents, and these communications may be sequential, e.g. to avoid wireless interference, leading to a slower convergence overall. In contrast, asynchronous methods require fewer communications, but more computation [2].

☆ This article is part of a Special issue entitled: 'TC 1.5 Networked systems (IFAC WC 2026)' published in Nonlinear Analysis: Hybrid Systems.

☆☆ This work was funded from SeaClear2.0, a project that received funding from the European Climate, Infrastructure and Environment Executive Agency (CINEA) under grant agreement No 101093822.

* Corresponding author at: Technical University of Cluj–Napoca, Romania.
E-mail addresses: tudor.santejudean@aut.utcluj.ro (T. Sântejudean), vineeth.satheeskumar-varma@univ-lorraine.fr (V.S. Varma), constantin.morarescu@univ-lorraine.fr (I.C. Morărescu), lucian@busoniu.net (L. Buşoniu).

https://doi.org/10.1016/j.nahs.2026.101750
Received 24 October 2025; Received in revised form 19 February 2026; Accepted 14 May 2026

To improve the trade-off between communication and computation, we propose Distributed Asynchronous Potential function Decrease (DAPD), which generalizes the Distributed Asynchronous Gradient Descent (DAGD) method of [2] in two major ways. Firstly, whereas DAGD runs Round-Robin (RR) updates of the agents in turn, DAPD leverages the graph structure of the agent interdependencies in the cost functions, together with a novel update scheduler that accelerates convergence and works as follows. At each DAPD iteration, the next player to update in each neighborhood is the one with the largest cost decrease at the previous update. This scheduling mechanism works in a decentralized fashion and allows parallel updates between non-neighboring players, thus reducing both communication and computation.

A second generalization over [2] is in the optimization settings addressed. While the first setting in which (*a*) the cost functions of the agents are assumed to be twice-differentiable, convex, and with Lipschitz-continuous first derivatives, was also tackled in [2], we additionally study here a less restrictive setting in which (*b*) the assumptions on the cost functions are loosened to Lipschitz continuity. We call (*a*) the Differentiable Setting (DS), and (*b*) the Lipschitz Setting (LS). In DS, we use gradient descent [11], whereas in LS we use a Lipschitz-based global optimizer called Simultaneous Optimistic Optimization (SOO) [12]. This is novel, since most works on distributed NE seeking require the more restrictive DS, among other assumptions like e.g., strong convexity [3,13,14]. We focus on pure NE seeking where agents have access to the information required to compute their cost function, when either (*a*) or (*b*) hold true, and aim to develop algorithms with convergence and stability guarantees, together with strong empirical performance. Theoretical analysis guarantees convergence to a pure NE in DS and to an ε-NE in LS, while the set of pure NE in DS is uniformly globally asymptotically stable. Numerical simulations show better performance when DAPD uses our novel scheduler than when using RR, along with a better computation-communication trade-off compared to baseline synchronous and asynchronous methods [2,9].

Additional related work studied the convergence of approximate asynchronous best-response dynamics for certain types of games, e.g. interference games [15], often employed in wireless communication. Stochastic algorithms are used by [16] to approximate the NE, while [17] use asynchronous better-response dynamics to converge to a NE in aggregative games. However, most of these papers on game theory are not concerned with the computational efficiency or convergence time of their algorithms and typically employ a RR protocol. The issue of communication efficiency has inspired several works on multi-agent event-triggered communication, see [18,19] that address control and optimization respectively. In contrast to these event-based methods, we focus here on distributed *asynchronous* optimization with the goal of pure NE seeking.

Next, Section 2 introduces the background on potential games and defines the problem. Section 3 presents the algorithmic contribution: the graph-based DAPD algorithm and update scheduler, followed by proofs of NE convergence and stability guarantees in Section 4. Numerical results are provided in Section 5. Section 6 concludes.

2. Preliminaries

Let n and a be strictly positive integers, i.e. $n, a \in \mathbb{Z}_+^*$. Consider the game $G := \{N, \{X_i\}_{i\in N}, \{J_i\}_{i\in N}\}$, where set $N := \{1, \dots, n\}$ represents the players (agents), and $x_i \in X_i$ denotes the action of player $i \in N$, with $X_i \subset \mathbb{R}^a$ a compact set. A complete action profile (strategy) of all the agents is denoted by $x := (x_1, \dots, x_n) \in X_1 \times \cdots \times X_n =: X \in \mathbb{R}^{n\times a}$. Each player $i \in N$ has a cost function (utility) $J_i : X \to \mathbb{R}_+$, and a set of neighbors $N_i \subseteq N$, where J_i only depends on the actions of agents in N_i. We also define the closed neighborhood of agent i, $\overline{N_i} := \{i\} \cup N_i$, the set containing agent i and all its neighboring players. Agent i can communicate to all agents $j \in N_i$. Using these neighborhoods we define the undirected cost graph $\mathcal{J} := \{(i, j) \mid i \in N_j \text{ and } j \in N_i; i, j \in N\}$.

Let $S = \{s_1, \dots, s_l\} \subset N$, $l \leq n$, and its complement w.r.t. N be $-S := N \setminus S$. We define the partial action profile containing all actions of the agents with indices in S as $x_S := (x_{s_1}, \dots, x_{s_l}) \in X_{s_1} \times \cdots \times X_{s_l} =: X_S$, and its complementary action profile $x_{-S} \in X_{-S}$, where the full action profile can be expressed as $x = (x_S, x_{-S})$. If $S = \{i\}$ is a singleton, i.e. $x_S = x_i$ for some $i \in N$, the action profile that excludes agent i will be referred to as $x_{-i} := x_{-S} = (x_1, \dots, x_{i-1}, x_{i+1}, \dots, x_n)$, and we have $x = (x_i, x_{-i})$ [20].

Assumption 1. G is an ordinal potential game with potential function $\phi : X \to \mathbb{R}_+$, i.e., $\forall i \in N$, $x \in X$ and $x_i' \in X_i$:

$$J_i(x_i, x_{-i}) > J_i(x_i', x_{-i}) \Leftrightarrow \phi(x_i, x_{-i}) > \phi(x_i', x_{-i}). \tag{1}$$

Moreover, X is convex and ϕ is continuous on X.

The potential function offers a global view of the entire game, unlike the individual cost functions that capture each player's goals [21]. When a player alters its action, the potential function keeps the sign of the difference between the player's old and new cost [22]. In specific cases like weighted or exact potential games, changes in utility directly translate to a corresponding shift (weighted or exact) in the potential function [20,23]. Another important concept is the pure NE [24], defined below.

Definition 1 (*Pure Nash and ε-Nash Equilibrium*)**.** A strategy $x^* \subseteq X$ is said to be a pure ε-NE iff $\forall i \in N$ and $\forall x_i \in X_i$:

$$J_i(x_i^*, x_{-i}^*) \leq J_i(x_i, x_{-i}^*) + \varepsilon. \tag{2}$$

In particular, for $\varepsilon = 0$, $x^* \in X^*$ becomes a pure NE, where X^* denotes the set of all pure NE in X.

At NE no player has an incentive to unilaterally change their strategy, because doing so would worsen their outcome, i.e. increase the associated cost/potential functions. If an action profile minimizes ϕ, then it is a Nash equilibrium [25]. All potential games admit at least one pure NE [20,26], which is unique if the potential function is strictly convex [27].

With Assumption 1 always holding, we define two different settings for our potential game framework: DS and LS. In DS, the cost functions are assumed to be twice-differentiable, convex, and with Lipschitz-continuous first derivatives.

Assumption 2 (*DS*). For all $i \in N$, the cost function J_i is twice differentiable and convex w.r.t. $x_i \in X_i$. Moreover, all first derivatives $\nabla_i J_i$ are Lipschitz-continuous with a finite Lipschitz constant $L_i > 0$ that is available to each player $i \in N$.

LS works with a less restrictive assumption on the costs, as they are only required to be one-sided Lipschitz-continuous around at least one global optimum [12].

Assumption 3 (*LS*). For all $i \in N$ and a given $x_{-i} \in X_{-i}$, there exists a semi-metric $l_i^{X_{-i}} : X_i \times X_i \to \mathbb{R}$ such that for at least around one of its global minima $x_i^* \in \arg\min_{x_i \in X_i} J_i(x_i, x_{-i})$, i.e., $\forall x_i \in X_i$, we have:

$$J_i(x_i, x_{-i}) - J_i(x_i^*, x_{-i}) \leq l_i^{X_{-i}}(x_i, x_i^*). \tag{3}$$

A pure NE can be reached through iterative updates of player strategies. Two commonly used approaches are best response and better response [27]. Best-response minimizes the cost of a given player under fixed strategies of the other players [28].

Definition 2 (*Best-Response Dynamics*). For any player $i \in N$, we define the best response $f_i : X_{-i} \to \mathcal{P}(X_i)$ as:

$$f_i(x_{-i}) := \arg\min_{x_i \in X_i} J_i(x_i, x_{-i}), \tag{4}$$

where $x_{-i} \in X_{-i}$, and $\mathcal{P}(X_i)$ is the power set of X_i. Ties in (4) are broken arbitrarily.

In contrast, better-response dynamics allow players to simply improve their strategy by lowering their cost function, without necessarily achieving the minimum in (4) [17,27].

Definition 3 (*Better-Response Dynamics*). $F_i : X \to X_i$ is a better response applied by player $i \in N$, iff, $\forall x \in X$:

$$J_i(F_i(x), x_{-i}) \leq J_i(x). \tag{5}$$

The relation holds with equality iff $x_i \in f_i(x_{-i})$.

Both dynamics converge to pure NE under appropriate conditions [2], and can be applied synchronously (all players update simultaneously in parallel) [28] or asynchronously (players take turns to update) [2]. Since computing the exact best response can be challenging [22], the better response is often more practical, and we will use it in the sequel. The next Lemma is a consequence of Assumption 1 and Definition 3.

Lemma 1. *Let $S \subset N$ be a set of players such that $\forall i, j \in S$, $i \neq j$, $(i, j) \notin \mathcal{J}$, i.e. players in S are not neighbors; and $x = x_S \cup x_{-S} \in X$. Denote by $F_S(x)$ the action profile obtained after all agents in S apply $F_i(x)$. Then:*

$$\phi(F_S(x), x_{-S}) \leq \phi(x_S, x_{-S}). \tag{6}$$

The relation holds with equality iff $x_i \in f_i(x_{-i}), \forall i \in S$.

Proof. Since players in S do not influence each other's utilities, $\forall x_i' \in F_S(x)$ a better response of x_i, we have:

$$J_i(F_S(x), x_{-S}) = J_i(x_i', x_{-i}) \leq J_i(x_i, x_{-i}) = J_i(x_S, x_{-S}),$$

and using (1), this leads to the desired result (6). □

We next provide two specific update rules for better/best-response dynamics, one for DS and another for LS.

Best and better-response for the DS problem

Under Assumption 2 (DS), one way to perform action updates is by using projected gradient descent [27]. One step of this method is defined as $\hat{F}_i : X_i \times X_{-i} \times \mathbb{R}_+ \to X_i$ such that:

$$\hat{F}_i(x_i, x_{-i}, \gamma_i) := P_{X_i}(x_i - \gamma_i \nabla_i J_i(x_i, x_{-i})), \tag{7}$$

where $P_A(x) := \arg\min_{y \in A} \|x - y\|_2$ is the projection of point x on a non-empty set A, and γ_i is the step size applied to the ith component of the J_i gradient, $\nabla_i J_i(x_i, x_{-i})$, taken w.r.t. player i. [2] used $\gamma_i \in (0, L_i^{-1}]$, where $L_i \in (0, \infty)$ is the Lipschitz constant of $\nabla_i J_i$, given by:

$$L_i := \sup_{x \in X} \|\nabla_i^2 J_i(x_i, x_{-i})\|_2, \tag{8}$$

and $\nabla_i^2 J_i$ is the ith element of the main diagonal in the Hessian of J_i. L_i must be available in advance to each player $i \in N$. The next lemma on projected gradient descent was given by [2], and will be used in our subsequent proofs.

Lemma 2. *Let $x \in X$ and $\gamma_i \in (0, L_i^{-1}]$, where $i \in N$. Then, $\hat{F}_i(x_i, x_{-i}, \gamma_i)$ is a better response, in the sense that:*

$$J_i(\hat{F}_i(x_i, x_{-i}, \gamma_i), x_{-i}) \leq J_i(x), \tag{9}$$

with equality iff $\hat{F}_i(x_i, x_{-i}, \gamma_i) = x_i$, i.e. iff the gradient $\nabla_i J_i(x_i, x_{-i})$ is zero or x_i is at the boundary of X_i.

Algorithm 1 M-step projected gradient descent

Declare procedure: **GD**(x, i, J_i, M, L_i)
Input: full strategy x, updating player i, J_i and L_i
initialize updated strategy $x_i^+ = x_i$
for each $k = 1, \dots, M$ **do**
  $\gamma_i = \arg\min_{\gamma \in (0, L_i^{-1}]} J_i(\hat{F}_i(x_i, x_{-i}, \gamma), x_{-i})$
  if $\|\hat{F}_i(x_i^+, x_{-i}, \gamma_i) - x_i^+\|_2 = 0$ **then**
    break for
  end if
  $x_i^+ = \hat{F}_i(x_i^+, x_{-i}, \gamma_i)$
end for
return x_i^+

We provide the M-step gradient descent in Algorithm 1, where $M \in \mathbb{Z}_+^*$ is the maximum number of updates performed to a given strategy x_i. At each such update, an adaptive step size γ_i for the cost gradient is computed numerically using a line search method (e.g. backtracking, Goldstein, etc.) [11], and this step size is then used in (7); which is an improvement over the fixed-step method of [2]. The algorithm stops when the number of updates has been exhausted or when $x_i^+ \in f_i(x_{-i})$ (this happens when the gradient becomes zero or the boundary of X_i is reached). In all cases, x_i^+ is returned as the new strategy. Note that we are using the full action profile x when describing the algorithm to simplify presentation, even though only x_i and x_{N_i} are used to compute x_i^+ (recall that only neighboring players of $i \in N$ influence its utility J_i, and therefore the better response $\hat{F}_i$).

Note that a best-response is obtained by Algorithm 1 as $M \to \infty$ [2]. Moreover, by applying Lemmas 1 and 2 to Algorithm 1, we get the following new result.

Corollary 1. *Let $\hat{F}_i^m(x) := \hat{F}_i(\hat{F}_i^{m-1}(x), x_{-i}, \gamma_i)$, where i is the updating player and $m \in \mathbb{Z}_+$ is the number of projected gradient steps applied to x_i; for $m = M$, $\hat{F}_i^m(x)$ is the output of Algorithm 1. Then, $\forall m \in \mathbb{Z}_+^*$, $\hat{F}_i^m(x)$ is a better response:*

$$J_i(\hat{F}_i^m(x), x_{-i}) \le J_i(x_i, x_{-i}). \tag{10}$$

Moreover, due to Lemma 1, (10) can be extended to any set $S \subseteq N$ in which players do not influence each other's utilities, in the sense that $\forall i \in S$:

$$J_i(\hat{F}_S^m(x), x_{-S}) \le J_i(x_S, x_{-S}), \tag{11}$$

where $\hat{F}_S^m(x)$ is the action profile obtained by applying in parallel the better responses $\hat{F}_i^m(x)$ for all $i \in S$.

Proof. Lemma 2 proves the initial case when $m = 1$. By induction, $\hat{F}_i^m(x)$ is obtained $\forall m \in \mathbb{Z}_+^*$ by repeatedly applying the same Lemma. Finally, using Lemma 1 to generalize the result to any set S of players that do not influence each other's utilities, we get (11). □

Better-response for the LS problem

Under Assumption 3 (LS), a Lipschitz-based global optimizer can be used to implement better responses. We select SOO [12], and adapt it to minimize any cost function $J_i : X_i \to \mathbb{R}$ (which we then use for agent i's cost, hence the index). The optimization process works by hierarchically partitioning the domain X_i into a tree structure denoted by $\mathcal{T}$. Each node (i, d, j) in the tree is labeled by $X_{i,d,j} \subseteq X$ and by a point $x_{i,d,j} \in X_{i,d,j}$ (center of the subset, except at the root $(i, 0, 0)$, where we take a special value explained later), where $d \ge 0$ is the depth of the node in the tree and $j \ge 0$ is the node's index at a given depth. The expansion process of $X_{i,d,j}$ splits it into R smaller sets $X_{i,d+1,j'}$ (usually of equal size), where $j' = 1, \dots, R$, and the root label $X_{i,0,0}$ is equal to the entire domain X_i. The collection of leaves at depth d of the currently explored tree is denoted by $\mathcal{L}_d$.

At each iteration k, SOO expands at most one leaf per depth. We define $\mathcal{L}_{\le d}$ as the collection of leaf nodes with depths up to d. A leaf (i, d, j) is chosen for expansion if its value $J_i(x_{i,d,j})$ is minimal among all leaves in $\mathcal{L}_{\le d}$. Once the node (i, d, j) was expanded and leaves $(i, d+1, j')$ created, J_i is evaluated at the newly generated leaf centers $x_{i,d+1,j'}$. The algorithm incorporates a depth-limiting function $d_{\max}(k)$, chosen by default as $d_{\max}(k) = \sqrt{k}$, to manage the balance between deep and wide tree exploration. The algorithm is run until the budget M is exhausted, at which point it returns the sample corresponding to the lowest cost seen. We summarize SOO in Algorithm 2 [12]. Note that SOO does not explicitly use the specific value of the Lipschitz constant of J_i, nor the semi-metric in (3); however, J_i must still be Lipschitz-continuous, as this is a fundamental requirement of SOO.

In DAPD, the current strategy x_i is chosen as the representative sample of the root node $(i, 0, 0)$ associated with the full set X_i, i.e. $x_{i,0,0} = x_i$, which allows us to always generate solutions that are no worse than $J_i(x_i, x_{-i})$. For ease of reference, we define by $\tilde{F}_i : X_i \times X_{-i} \to X_i$ the result of agent i applying SOO as in Algorithm 2 to J_i. SOO guarantees ε-optimality, where ε decreases with larger expansion budget M [12], see also Theorem 3 later. Thus, it is easy to prove that $\tilde{F}_i(x)$ implements a (not necessarily strictly) better-response, and that Corollary 1 holds also for better-responses implemented via SOO.

Algorithm 2 SOO

Declare procedure: **SOO**$(x, i, X_i, J_i, M, R, d_{\max}(\cdot))$
Input: full strategy x, updating player i, solution space X_i, cost function J_i, budget M, branching factor R, depth function $d_{\max}(\cdot)$

1: initialize solution $x_{i,0,0} = x_i$, $\mathcal{T} = X_{i,0,0}$, $k = 1$
2: **loop**
3: $J_{\min} = \infty$
4: **for** $d = 0$ to $\min(\text{depth}(\mathcal{T}), d_{\max}(k))$ **do**
5: $(i, d, j^\dagger) \leftarrow \arg\min_{(i,d,j)\in\mathcal{L}_d} J_i(x_{i,d,j}, x_{-i})$
6: **if** $J_i(x_{i,d,j^\dagger}) \leq J_{\min}$ **then**
7: expand $(i, d, j^\dagger)$, by adding its R children as leaves to $\mathcal{T}$
8: $J_{\min} = J_i(x_{i,d,j^\dagger}, x_{-i})$
9: $k = k + 1$
10: **if** $k > M$ **then**
11: **break loop**
12: **end if**
13: **end if**
14: **end for**
15: **end loop**
16: return $\widehat{x^*} = \arg\min_{(i,d,j)\in\mathcal{T}} J_i(x_{i,d,j}, x_{-i})$

3. DAPD

This section presents the DAPD method, a graph-based extension of the DAGD algorithm used in [2]. One main contribution of the present paper over [2] is the generalization of DAGD to graph-based communication and cost updates. While [2] updated the strategy of a single agent at a time, DAPD will schedule multiple agent updates in parallel without ever updating neighbors in the graph at the same time. This latter condition is required because updating in parallel multiple agents in the same neighborhood leads to a (partially) synchronous behavior of DAPD, which may lead to lack of convergence [9].

The DAPD method is presented in Algorithm 3, where the procedure is applied in parallel (at the player level) for all players $i \in N$. Each player i starts from an initial strategy $x_i(0)$, an initial $\text{flag}_i = 1$ signifying its cost can still be improved, and an initial scheduler score b_i (line 1); scores b_i will be useful for the novel Largest Cost function Decrease (LCD) scheduler that we present afterwards. At each iteration k, each player verifies if the cost of any of the players in its closed neighborhood $\overline{N_i}$ can still be improved (line 3). In such a case, the scheduling mechanism is run by each player individually (line 4), so that no more than one player in each neighborhood makes an update. Any updating player i applies a better response in the form of gradient descent (Algorithm 1) or SOO-based updates (Algorithm 2) (lines 7–10) depending on the type of problem. Note that by a slight abuse of notation, instead of providing the full strategy like in Algorithms 1 and 2, to emphasize the decentralized nature of DAPD the optimizers are called with only the actions in the closed neighborhood $x_{\overline{N_i}}(k)$. If the normed difference between the old and the updated action drops below a preset tolerance τ, flag_i is set to zero to signal the cost J_i was optimized w.r.t. the actions of the players in its neighborhood N_i (lines 11–12). The scheduler score b_i is then updated (line 13). The new action $x_i(k+1)$, the value of flag_i and of the score b_i are transmitted to all neighboring players in N_i (line 14). The algorithm stops when the number of iterations $k_{\max}$ is exhausted, at which point DAPD returns the current strategy $x_i(k_{\max})$ as an approximation of its corresponding action component x_i^* of a pure NE (line 15).

The existing algorithm in [2] is a special case of DAPD, for which only the DS problem is considered with a RR scheduler, which updates the action of only one player per iteration. This scheduler sequentially visits all players, from 1 through n, resulting in an update sequence of length n. To keep the RR updates decentralized, the players' updating order must be fixed a priori. RR was proven to converge to a pure NE when using better responses [2]. While RR does not involve intricate computations or storage between iterations – e.g., Algorithm 3 running with RR does not need to store b_i [2] – it comes with a major drawback. To illustrate it, consider the case when $x_{-i} = x_{-i}^*$, but x is not yet a pure NE, i.e. $x_i \neq x_i^*$. Since RR periodically cycles through all players, DAPD will inevitably waste updates on players different from i that do not lead closer to the pure NE. To address this issue, the order of player updates can follow a ranking system. For example, based on the decrease of their cost function after the update occurred. Motivated by this intuition, we present our novel DAPD scheduler, called LCD.

The LCD scheduler stores the per-agent decreases in the cost function after the M-step projected gradient descent or the SOO-based action update was performed. Specifically, each updating player i stores $b_i := J_i(x_i(k), x_{N_i}(k)) - J_i(x_i(k+1), x_{N_i}(k))$, while non-updating players keep their last score value unchanged. The action update of a player i is triggered only if, among all players in its closed neighborhood $\overline{N_i}$, it has the largest cost decrease stored, i.e. $i = \max\{\arg\max_{j\in\overline{N_i}} b_j\}$; in case multiple players share the maximum cost decrease, the one with the largest index will update. Note that at the start of DAPD all scores b_i will be initialized to infinity, i.e. $b_i = \infty, \forall i \in N$, to ensure that each player is initially updated at least once.

A possible issue with LCD is when following an update, player i reaches its best response w.r.t. the other players' actions, but x is not yet an NE (i.e. $x_i \in f_i(x_{-i})$, but $x_i \neq x_i^*$). In this case, player i might not update again unless all other players have reached

Algorithm 3 DAPD

Input: players N, graph $\mathcal{J}$, cost functions J_i, $\forall i \in N$,
initial strategy $x(0) \in X$, maximum number of iterations $k_{\max}$, tolerance τ
Input for DS: GD steps M, constants L_i, $\forall i \in N$
Input for LS: SOO budget M, $R, d_{\max}(\cdot)$
In parallel: (applied by each player $i \in N$)
initialize $x_i(0)$, $\text{flag}_i = 1$, and scheduler score $b_i = \infty$
for each iteration $k = 0, 1, \dots k_{\max} - 1$ **do**
  if $\sum_{j \in \overline{N_i}} \text{flag}_j > 0$ **then**
    get updating player $\theta \leftarrow \max\{\arg\max_{j \in \overline{N_i}} b_j\}$
    if $i = \theta$ **then**
      $\text{flag}_i = 1$
      if problem type is DS **then**
        $x_i(k+1) \leftarrow \textbf{GD}(x_{\overline{N_i}}(k), i, J_i, M, L_i)$
      else (problem type is LS)
        $x_i(k+1) \leftarrow \textbf{SOO}(x_{\overline{N_i}}(k), i, X_i, J_i, M, R, d_{\max})$
      end if
      if $\|x_i(k+1) - x_i(k)\|_2 < \tau$ **then**
        $\text{flag}_i = 0$
      end if
      $b_i = J_i(x_i(k), x_{N_i}(k)) - J_i(x_i(k+1), x_{N_i}(k))$
      communicate $x_i(k+1)$, flag_i, and b_i to all neighbors N_i
    end if
  end if
end for
return $\widehat{x_i^*} = x_i(k_{\max})$.

their best responses. The latter might never happen if given the current strategies, an updating player reaches its best response only asymptotically, in which case x_i will never converge to x_i^*. To avoid this, after every $p \in \mathbb{Z}_+^*$ iterations of DAPD run with the LCD scheduler, an RR update sequence of length n will be triggered (not shown in the pseudocode); DAPD can thus be seen as having an overall update sequence length of $p+n$. These regular RR updates serve as a fallback mechanism for when LCD updates stagnate in the sense from the above edge case. Later in our analysis, we leverage this mechanism to establish convergence guarantees toward the NE.

We will now discuss the tuning of parameter p. If p were chosen 0, DAPD would only run with RR, which would often be slower than LCD, as will be illustrated in the experiments of Section 5. Conversely, when $p \to \infty$, DAPD only runs with LCD, which might lead to stagnation and lack of convergence as explained above. Both these limiting cases are undesirable, which is why we require $p \in \mathbb{Z}_+^*$. In practice, we suggest taking p much larger than the length n of the RR update sequence.

4. Convergence analysis

This section will study the convergence and stability of DAPD. More challenging to derive than in [2], these theoretical results must be studied in the context of graph-based communication and computation, and also proven for the newly proposed LCD scheduler. Another key contribution of our work is the study of ϵ-NE convergence under the less restrictive LS conditions.

Theorem 1 (*Convergence in DS*). *Subject to Assumptions 1 and 2 (DS), DAPD running with either the RR or LCD schedulers and Algorithm 1 for better-response calculation guarantees convergence to a pure NE.*

Proof. We first prove the convergence of the DAPD version with RR and then with LCD. Let $x(k)$ be the action profile, and $\Theta(k)$ be the updating players chosen by RR at iteration $k \in \mathbb{Z}_+^*$. Using Corollary 1, we get:

$$J_i(\hat{F}^M_{\Theta(k)}(x), x_{-\Theta(k)}) \leq J_i(x_{\Theta(k)}, x_{-\Theta(k)}), \tag{12}$$

with equality iff $\hat{F}_i^M(x) = x_i, \forall i \in \Theta(k)$, i.e. all gradients computed in Algorithm 1 at step k are zero. From (12), it quickly follows that $\phi(x(k))$ is decreasing. We distinguish two possible cases: (i) $\exists k' \in \mathbb{Z}_+^*$ such that $\forall k > k'$, we have: $\hat{F}_i^M(x(k+l)) = x_i(k+l)$, $\forall i \in \Theta(k)$, and $\forall l \in \mathbb{Z}_+$ with $0 \leq l < n$, where n is the length of the updating period of RR; and (ii) $\nexists k' \in \mathbb{Z}_+^*$ with the above property.

In case (i), all projected gradients remain stationary across a full RR update period. Thus, $x(k) = \cdots = x(k+n-1)$, and since the RR scheduler updates all players each n iterations, $\hat{F}_i^M(x(k+l)) = x_i(k+l)$, $\forall i \in N$ and $0 \leq l < n$. By means of induction, one can prove that future projected gradients remain stationary, i.e. the strategy $x(k)$ remains unchanged $\forall k > k'$. Using the convexity of the utilities, we have that $J_i(x_i, x_{-i}) = \min_{x_i'} J_i(x_i', x_{-i})$ for all $i \in N$, i.e. $x(k) = x^*$ per Definition 1.

In case (ii), within a full RR update period, all players are updated at least once, and $\exists i \in N$ s.t. $J_i(\hat{F}_i^M(x), x_{-i}) < J_i(x)$. Per inequality (12), this guarantees a strict decrease in the potential function after each n iterations of DAPD, i.e. $\phi(x(k+n)) < \phi(x(k)), \forall k \in \mathbb{Z}_+$. Furthermore, ϕ is continuous on the compact set X and the set of pure NE is the invariant set of the updates. Thus, by applying the LaSalle invariance principle for discrete-time systems, see Theorem 1 in [29], the potential function will asymptotically converge toward a pure NE.

Next, we show the convergence to pure NE of DAPD with LCD. Recall that for each p DAPD iterations, an RR updating sequence is triggered (following the RR scheduler approach). This leads to a strict decrease in the potential function every $p+n$ DAPD iterations, unless a pure NE is reached. The convergence is evident per the argument in the second case of the RR proof above. □

Further we will prove that in DS, the set of pure NE of the ordinal potential game G is uniformly globally asymptotically stable (UGAS) w.r.t. DAPD. This guarantees that once the pure NE was reached, even if players' strategies experience perturbations or deviations, the dynamics of the game ensure that they will return to equilibrium [30]. We first interpret DAPD as a dynamic system described by the equations:

$$\begin{aligned} x_{\Theta(k)}(k+1) &= \hat{F}^M_{\Theta(k)}(x(k)) \\ x_{-\Theta(k)}(k+1) &= x_{-\Theta(k)}(k), \end{aligned} \tag{13}$$

where $\Theta(k)$ represents the updating players at iteration k triggered with any of the schedulers proposed above.

Theorem 2 (*Stability in DS*). *The set of pure NE of the game G is UGAS for dynamics (13), provided the potential function ϕ is convex.*

Proof. In Theorem 1 (DS), we proved that as long as a pure NE is not reached, ϕ strictly decreases over a period of $p+n$ iterations for all proposed schedulers. By using the convexity of ϕ and the proof line of Theorem 2 in [2], it follows that the attractor set $\{(x, \Theta) | x \in X^*, \Theta \subset N\}$ is UGAS. □

We provide next the convergence result for the LS problem.

Theorem 3 (*Convergence in LS*). *Subject to Assumptions 1 and 3 (LS), DAPD running with either the RR or LCD schedulers and Algorithm 2 for better-response calculation guarantees convergence to a pure ε-NE.*

Proof. We identify two possible cases for convergence in LS. In the first case, $\exists k'$ for which $\forall k > k'$ and $\forall i \in N$, $J_i(x(k)) = J_i(x(k'))$, i.e. after k' iterations of DAPD, the cost functions cannot be decreased anymore by applying SOO-based better-responses $\tilde{F}$ with Algorithm 2. This may happen since SOO only provides ε-closeness to optimality given a finite expansion budget M. Once the costs and agent strategies remain stationary (i.e., for $k > k'$), we denote by $\varepsilon_i > 0$ the closeness to optimality for each player i, which due to Corollary 4.3 in [12], satisfies:

$$\varepsilon_i = J_i(x_i, x_{-i}) - J_i(x_i^*, x_{-i}) = \begin{cases} O\left(\sqrt{M}^{-1/\beta_i}\right) & \text{if } \beta_i > 0 \\ O\left(\gamma_i^{\sqrt{M}/C_i}\right) & \text{if } \beta_i = 0 \end{cases}$$

where $O(\cdot)$ is the Bachmann–Landau notation, $d_{\max}(k) = \sqrt{k}$, $\gamma_i \in (0,1)$ and $C_i \geq 1$ are constants related to the SOO partitioning, and $\beta_i \geq 0$ represents the problem complexity, measured by the near-optimality dimension. By taking $\varepsilon = \max_i \varepsilon_i$, we get:

$$J_i(x_i, x_{-i}) - J_i(x_i^*, x_{-i}) \leq \varepsilon, \forall i \in N,$$

which means that players reached an ε-NE by definition.

In the second case, no k' exists such that $\forall k > k'$, all costs $J_i(x(k))$ remain constant. Thus, we will have a strict decrease in the potential function after each n iterations of DAPD running with RR, or after each $n+p$ iterations of DAPD. This translates to $\phi(x(k+n+p)) < \phi(x(k)), \forall k$. As the invariant set associated with SOO updates is the set of ε-NE, and because ϕ is continuous on the compact set X and strictly decreasing after each $n+p$ iterations, DAPD converges to an ε-NE, owing again to the LaSalle invariance principle for discrete-time systems [29]. Note that this convergence is asymptotic, and not guaranteed to occur in a finite number of steps k' as in the first case above. □

Remark 1. Parameter ε does not directly depend on the number of agents, but on the computation budget M and the near-optimal dimension β_i of each player's cost function. This is because we apply SOO to minimize the cost J_i of player i only w.r.t. its own action x_i, while keeping the actions of all other (neighboring) players x_{-i} fixed.

5. Numerical results

5.1. Setup

We consider the problem of distributed formation control, in which a set of agents must reach a predefined formation (possibly in addition to other goals), and use the potential game framework to tackle it. Consider $n \in \{4, 9, 16, 25\}$ agents described by positions

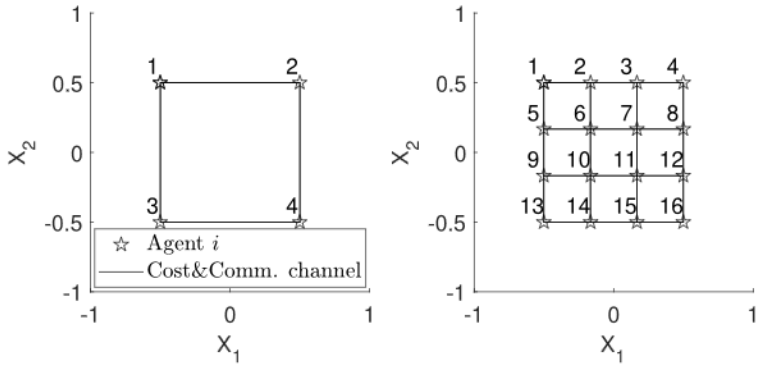

Fig. 1. Grid-like agent formations and cost/communication graphs used in the experiments. Left: 4 agents; right: 16 agents. Each agent is marked with a black star, while the black edges represent the cost and communication channels between neighboring agents.

(taken to be the actions in the game) $x_i \in X_i := [-1, 1]^2$. To achieve the desired formations, agents must be placed on a square-like grid of side length 1, in which the discretization step across each dimension equals $\frac{1}{\sqrt{n}-1}$ ($\sqrt{n}$ agents on each grid axis); Fig. 1 gives two examples for $n = 4$ (left) and $n = 16$ (right). Any formation can be described by equations $x_i - x_j = d_{ij}$, with $d_{ij} = -d_{ji}, \forall i, j \in N$; e.g. of distances for the first formation ($n = 4$) in Fig. 1: $d_{12} = [1; 0]$, $d_{23} = [-1; -1]$, etc.

The graph $\mathcal{J}$ is connected and undirected, linking every pair of neighboring agents on the grid; see again Fig. 1, in which the lines are graph edges. Agents update their positions based on local information received from their neighbors in the graph; e.g., in the 4-agent formation $N_1 = \{2, 3\}$, i.e. players 2 and 3 are neighbors of 1.

We provide first numerical results for the DS problem, followed by results for LS.

5.2. DS problem

We define the cost function of any agent $i \in N$ by:

$$J_i(x) = r_i \left(\exp\left(\|x_i - s_i\|_2^2\right) - 1\right) + \sum_{j \in N_i} \left(\exp\left(\|x_i - x_j - d_{ij}\|_2^2\right) - 1\right), \tag{14}$$

where the first term is related to the control effort (energy cost) of agent i, indirectly evaluated by the distance between its initial position $s_i \in X_i$ and x_i, and r_i is the weight associated with it. The second term is the cost associated with the deviation from formation, and becomes zero when agent i is in the desired formation with all of its neighbors N_i.

Next, we define a potential function of game G:

$$\phi(x) = \sum_{i=1}^{N} \left(r_i \left(\exp\left(\|x_i - s_i\|_2^2\right) - 1\right) + \frac{1}{2} \sum_{j \in N_i} \left(\exp\left(\|x_i - x_j - d_{ij}\|_2^2\right) - 1\right) \right). \tag{15}$$

Using the fact that $d_{ij} = -d_{ji}, \forall i, j \in N$, it can easily be proven that $\forall i \in N$, $x \in X$ and $x_i' \in X_i$, we have:

$$J_i(x_i, x_{-i}) - J_i(x_i', x_{-i}) = \phi(x_i, x_{-i}) - \phi(x_i', x_{-i}), \tag{16}$$

i.e. ϕ is an exact potential function. All exponents are quadratic in x_i, which means both J_i and ϕ are convex and two times differentiable w.r.t. any $i \in N$, validating Assumptions 1–2.

Experiments in DS are organized as follows. We first compare numerically the RR and LCD schedulers when used with the asynchronous methods DAPD and Asynchronous Best-Response Dynamics (ABRD) [2]. With the best performing schedulers, we then pit DAPD and ABRD against a synchronous baseline, Distributed Synchronous Gradient Descent (DSGD) [9]. Note that ABRD uses the best-response dynamics of (4) to update: $x(k+1) = (f_{\Theta(k)}(x(k)), x_{-\Theta(k)}(k))$, while DSGD applies in parallel at each iteration a single step of gradient descent to all players: $x(k+1) = P_X(x - \gamma \nabla \phi(x(k)))$; see [2] for more details.

The parameters used in all experiments are: effort weights $r_i = 10^{-4}$, tolerance $\tau = 10^{-4}$, gradient updates $M = 5$, and $L_i = 10^2, \forall i \in N$. Since $\gamma_i \in (0, L_i^{-1}]$, we choose the adaptive gradient step $\gamma_i \in (0, 10^{-3}]$ for DAPD and ABRD, while DSGD will run with a fixed step size of 10^{-3} (in DSGD, agents need to agree on the step size before the algorithm runs in order to keep the updates decentralized).

In our first batch of experiments, performed with DAPD and ABRD on the grid-like formations from above, we compared RR and LCD in terms of iterations to convergence. We conducted 25 runs for each scheduler, with agents starting from uniformly random positions in X, which are the same for all schedulers in any given run. Fig. 2 gives the mean number of iterations till convergence, with 95% confidence bounds on the mean indicated by the colored area. Overall, LCD is statistically better than RR for both DAPD and ABRD, with the difference increasing with the number of agents n. An important conclusion is that our LCD scheduler improves not only the convergence speed of DAPD, but also of other distributed asynchronous methods like ABRD.

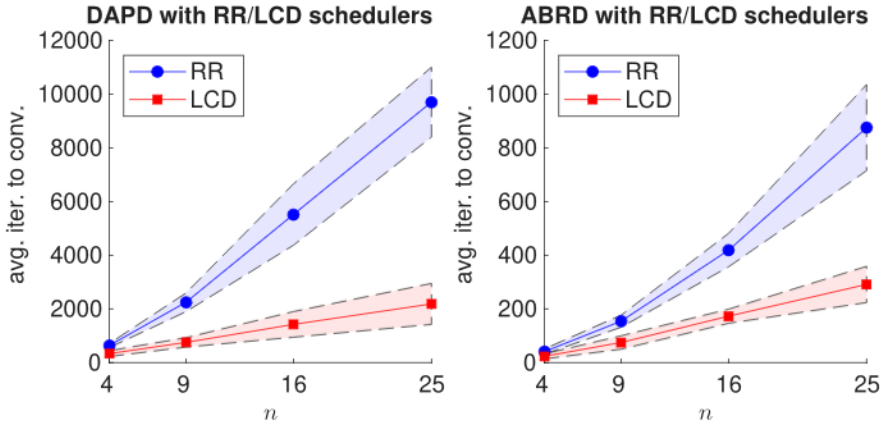

Fig. 2. Comparison between DAPD (left) and ABRD (right) running with RR and LCD schedulers in DS. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

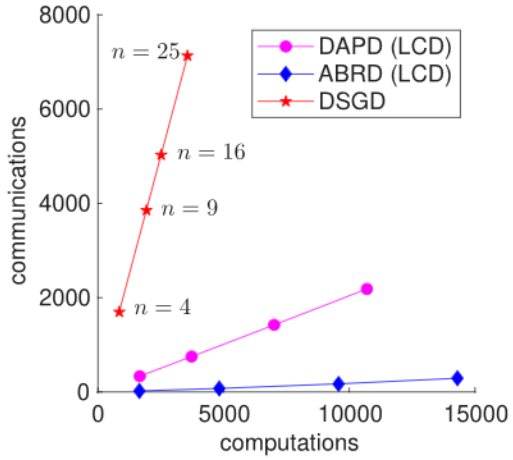

Fig. 3. Comparison between DAPD, DSGD and ABRD in DS, in which the asynchronous methods are run with the LCD (DAPD, ABRD) scheduler. Results in terms of computation and communication units till convergence are reported on average across 25 experiments, and markers sequentially give the results with formations of 4, 9, 16 and 25 agents.

In the second batch of experiments we compared DAPD with LCD against DSGD and ABRD, the latter using LCD. Results are given as communication and computation counts till convergence; in which any number of computations/communications performed in parallel at step k are only counted once. Each gradient-descent step applied by an updating player $i \in \Theta(k)$ in Algorithm 1 represents a computation unit, leading to a maximum of M such units per iteration for DAPD. The computation count per iteration for DSGD is 1, while for ABRD it is the maximum number of gradient steps applied by any of the updating players until reaching their best response. Communication counts for DAPD and ABRD will be 1 per iteration, while for DSGD it is 2, assuming players agree on the order in which they communicate (e.g. firstly, the odd players of the odd grid lines and the even players of the even grid lines communicate in parallel, while the remaining players communicate in the second communication step).

Fig. 3 shows the mean results across 25 experiments performed on all grid-like formations. For scenarios with large communication costs and small computation costs, ABRD is preferred. Conversely, DSGD is more efficient when computation costs outweigh communication costs. DAPD provides a balanced middle ground.

5.3. LS problem

We design new cost functions incorporating a non-convex term Q based on radial basis functions:

$$Q(x_i) = 1 - \max\left\{ h_1 \exp\left[-\sum_{j=1}^{2} \frac{(x_{i,j} - c)^2}{b^2}\right]; h_2 \exp\left[-\sum_{j=1}^{2} \frac{x_{i,j}^2}{b^2}\right]\right\}, \tag{17}$$