

Dynamic opponent modelling in fictitious play

MICHALIS SMYRNAKIS¹ AND DAVID S. LESLIE¹

¹ *Department of Mathematics University of Bristol UK*
Email: m.smyrnakis@bris.ac.uk, david.leslie@bris.ac.uk

Distributed optimisation can be formulated as an n player coordination game. One of the most common learning techniques in game theory is fictitious play and its variations. However fictitious play is founded on an implicit assumption that opponents' strategies are stationary. In this paper we present a new variation of fictitious play in which players predict opponents' strategy using a particle filter algorithm. This allows us to use a more realistic model of opponent strategy. We used pre-specified opponents' strategies to examine if our algorithm can efficiently track the strategies. Furthermore we have used these experiments to examine the impact of different values of our algorithm parameters on the results of strategy tracking. We then compared the results of the proposed algorithm with those of stochastic and geometric fictitious play in three different strategic form games: a potential game and two climbing hill games, one with two players and the other with three players. We also tested our algorithm in two different distributed optimisation scenarios, a vehicle target assignment game and a disaster management problem.

Our algorithm converges faster to the optimum than both the competitor algorithms in the strategic form games and the vehicle target assignment game. Hence by placing a greater computational demand on the individual agents, less communication is required between the agents.

In the disaster management scenario we compared the results of particle filter fictitious play with the ones of Matlab's centralised algorithm `binprog` and the centralised preplanning algorithm of [1]. In this scenario our algorithm performed better than the preplanning algorithm in two of the three performance measures we used.

Received 00 Month 2004; revised 00 Month 2004

1. INTRODUCTION

Disaster management refers to a process that is designed to prevent, reduce the impact, respond or recover from a natural or manmade disaster, like earthquakes, fires or floods. In these cases the resources that are available should be allocated in an optimal way to minimise the losses either in terms of human lives or damaged properties and environment [2].

A centralised solution is infeasible for many different reasons. One is restrictions in the communications between the emergency units. It is highly probable that after the disaster there will be a restriction in the communication range of the emergency units, and thus they will be able to communicate and exchange information only with other units that are near them. A second is that these situations are of large scale and of high complexity. The demand of computational power and the amount of time that is needed to find an

optimal solution makes the results inefficient. Taking into account these difficulties it seems necessary to find a way for the emergency units to coordinate by themselves without the help of a central decision maker.

Thus algorithms for distributed optimisation are needed. Besides disaster management [3], these algorithms are also useful in applications such as sensor networks [4], traffic control [5] and scheduling problems [6] where centralised approaches are inefficient.

Distributed optimisation cannot be applied in every optimisation task. When an optimisation task of dimensionality k and of utility function u_g , which is the function we want to optimise, can be defined as k different subproblems with a utility function u_d , where $d = 1, \dots, k$, then we can solve it using distributed optimisation algorithms. It is possible for the different subproblems to be related and thus the different u_d 's will have some common features. Each

of the sub-problems can be solved separately and u_g is an aggregation of the separate u_d 's [7, 8]. Distributed optimisation algorithms can be used to propose a solution for each subproblem taking into account the relationships between the different utility functions u_d 's. Then the solution will be the combination of the solutions of each sub-problem.

A class of algorithms that are used in distributed optimisation tasks are pre-planning algorithms. These algorithms solve the problem in advance using a centralised method, when time and computational power is not restrictive as in the real scenario. Then the results that have been acquired in advance are used to obtain a solution in the real task. An example of this approach is [1], where a disaster management scenario is solved. A centralised random neural network is trained in advance, and when the disaster occurs each individual independently uses the fitted network to decide which action to take. These methods have small communication cost since the agents should exchange once the information which is necessary as input for the pre-planning algorithm. When the real situation is similar to the ones that were used to train the pre-planning algorithm these algorithms result in solutions which are approaching the global optimum. But in cases where either the scenarios are continuously changing and it is necessary for the algorithms to adapt to the new environment, or the scenarios that were used to train the algorithm are different from the real one, then these algorithms are not efficient.

On the other hand there are algorithms that provide distributed solutions. A category of these algorithms is the distributed complete algorithms. This category includes algorithms like Dynamic Programming Optimisation (DPOP) [9], Asynchronous Distributed Optimisation (ADOPT) [10] and Optimal Asynchronous Partial Overlay (OptAPO) [11]. The solution of these algorithms approach the global optimum but they are intractable for large scale scenarios or very complicated problems. This is because these algorithms require the agents to exchange complicated data structures, that are costly to be transmitted to everyone in large scale problems. Also the memory that is required by some of them increases exponentially with the complexity of the problem.

Message passing algorithms are another class of distributed optimisation algorithms with the canonical example being the Max-Sum algorithm [12]. In contrast to distributed complete algorithms, algorithms that belong in this class result in agents that iteratively exchange data structures locally. This reduces the cost of communication and also the complexity of the problem each agent has to solve, since each agent only solves a sub-problem that involves the agents that are in his range. The solutions of message passing algorithms can be less efficient than the distributed complete algorithms in some problem scenarios.

Iterative algorithms based on best response such as regret matching [13], the maximum gain algorithm [14] and fictitious play [15] are another category of distributed optimisation algorithms in which the problem is cast as a game. They require less communication than the other two categories described above, since their only requirement is that agents should inform the others about their current state. Since these algorithms use less information than the message passing and distributed complete algorithms they are more likely to get stuck in local optima. Nevertheless these algorithms are tractable when there are severe restrictions on communication. They are also robust to situations in which some agents do not follow the correct algorithm.

Fictitious play is an iterative algorithm that can be used to solve games. Under fictitious play each player maintains some beliefs about his opponents' strategies, and based on these beliefs he chooses the action that maximises his expected reward. The players then update their beliefs about their opponents' strategies after observing the actions of their opponents. There are theoretical proofs that fictitious play converges to Nash equilibrium for certain kind of games, but in practice this convergence can be very slow [16, 17]. This is because it implicitly assumes that the other players use a fixed strategy in the whole game.

In this paper we introduce a variation of fictitious play in which, in contrast with the classic algorithm, players do not assume that their opponents have fixed strategies. Therefore they attempt to track other players' strategies using particle filters. We observe that this approach reduces the number of steps that fictitious play takes to converge, and hence the communications overhead between the distributed optimisers that are required to find a solution to the distributed optimisation problem.

The remainder of this paper is organised as follows. We start with a brief description of game theory, different categories of games, fictitious play and stochastic fictitious play. Section 3 introduces a particle filter based algorithm to update the beliefs about opponents' strategy and the results of a tracking scenario. Section 4 presents the results of the method described in Section 3 when incorporated in stochastic fictitious play, comparing with the results of classic stochastic fictitious play [16] and geometric stochastic fictitious play [18]. Sections 5 and 6 include the description and the results for the vehicle target assignment game and the disaster management simulation scenario respectively. We finish with a conclusion.

2. GAMES

2.1. Game Theory

Game theory is the field of science that deals with the way someone should take a decision when there

is interaction between the decision makers. Games can be divided into two general categories: strategic form games and extensive form games. The main difference between strategic and extensive form games is that in strategic form games the actions are made simultaneously but in extensive form games the decision makers might act with knowledge of the action of their opponents. For that reason a basic element of extensive form games is the order of the moves. Generally we can say that extensive form games are a form of multi-player decision trees. The rest of this paper will focus on strategic form games since they map more naturally to the distributed optimisation framework.

The elements of a strategic form game are [19]

- a set of players $i \in 1, 2, \dots, I$,
- a set of actions $s^i \in S^i$ for each player i ,
- a set of joint actions, $s = (s^1, s^2, \dots, s^I) \in S^1 \times S^2 \times \dots \times S^I = S$,
- the payoff function $u^i : S \rightarrow \mathbf{R}$ for each player i ,

where $u^i(s)$ is the utility that player i will gain after a specific joint action s has been played. We will often write $s = (s^i, s^{-i})$, where s^i is the action of Player i and s^{-i} is the joint action of Player i 's opponents. The rules that the players use to select the action to be played in the game are called strategies. When an action of Player i is selected from his set of available actions using a deterministic rule then the player acts according to a pure strategy. In the case that he chooses an action based on a probability distribution σ^i over the available actions he acts according to a mixed strategy. We will use $\sigma = (\sigma^1, \dots, \sigma^I)$ to denote a joint mixed strategy and will often write $\sigma = (\sigma^i, \sigma^{-i})$ analogously to $s = (s^i, s^{-i})$. We will denote the expected utility a player i will gain if he chooses a strategy σ^i (resp. s^i), when his opponents choose the joint strategy σ^{-i} as $u^i(\sigma^i, \sigma^{-i})$ (resp. $u^i(s^i, \sigma^{-i})$).

Different kinds of strategic form games can be classified according to their payoff functions. In particular we separate games into coordination and non-coordination games. Two player zero-sum games are a class of non-coordination games where $\sum_i u^i(s) = 0$. In this kind of game when Player i obtains some utility u^i his opponent obtains a utility $-u^i$. Another class of non-coordination games are the constant sum games where $\sum_i u^i(s) = c$, where c is a constant. Generally we can characterise non-coordination games as competitive games since the utility that a player gains is his opponent's loss. Although non-coordination games are the most-studied games, they are the least applicable to distributed optimisation.

On the other hand in coordination games the players' payoffs are simultaneously maximised and agents must collaborate to choose the action that maximises their payoffs. A subcategory of coordination games are potential games. In a potential game utilities follows the formula:

$$u^i(s^i, s^{-i}) - u^i(\tilde{s}^i, s^{-i}) = \phi(s^i, s^{-i}) - \phi(\tilde{s}^i, s^{-i}) \quad (1)$$

where ϕ is a potential function and the above equality stands for every player i , for every action $s^{-i} \in S^{-i}$, and for every pair of actions $s^i, \tilde{s}^i \in S^i$, where S^i and S^{-i} represent the set of all available actions for Player i and his opponents respectively. Ordinal potential games provide us with a more relaxed connection of the utility and the potential function ϕ which is:

$$u^i(s^i, s^{-i}) - u^i(\tilde{s}^i, s^{-i}) > 0 \Leftrightarrow \phi(s^i, s^{-i}) - \phi(\tilde{s}^i, s^{-i}) > 0. \quad (2)$$

Considering again the distributed optimisation problem, one of the desired properties that the utilities of each subproblem should have is that their association with the global utility should be monotonic. In particular this implies that an action which improves or reduces the utility of a subproblem should respectively increase or reduce the global utility. It is not necessary that the change in the local utility should be the same as the one in the global utility. If we denote as $u_d(s)$ the utility that it is related with the d^{th} subproblem after a set of actions s has been chosen by the agents and $u_g(s)$ the global utility the set of actions s , then we can formulate this property as:

$$u_d(s) - u_d(\tilde{s}) > 0 \Leftrightarrow u_g(s) - u_g(\tilde{s}) > 0 \quad (3)$$

If we choose an appropriate form of the global utility then it can act as a potential function and so the relation between the local utility, $u_d(s)$, and the global utility, $u_g(s)$, as it is described in (3), becomes the same with the one of the ordinal potential games. Thus we can represent a distributed optimisation task as a multi-player potential game.

Wonderful Life Utility (WLU) is such a utility function. It was introduced in [20] and applied in [21] to formulate distributed optimisation tasks as ordinal potential games. Furthermore it has been proved [22] that also Distributed Constraint Optimisation Problems (DCOPs) can be formulated as ordinal potential games.

So the task of finding an optimum in a distributed optimisation problem can be seen as a search for a Nash equilibrium in a game. A set of pure strategies satisfying the property that, when the players choose these actions none of them can do better by unilaterally changing his or her strategy, is called a pure strategy Nash equilibrium [23]. In general we can define the Nash equilibrium as a joint mixed strategy $\hat{\sigma}$ that for each player i satisfies

$$u^i(\hat{\sigma}^i, \hat{\sigma}^{-i}) \geq u^i(s^i, \hat{\sigma}^{-i}) \quad \text{for all } s^i \in S^i \quad (4)$$

Every potential game has at least one pure strategy Nash equilibrium [17] (there may be more than one). A sub case of the pure equilibria are the strict equilibria. A pure equilibrium is strict if each player at the equilibrium has a unique best response to his opponents actions.

An important category of equilibria which are of particular interest in distributed optimisation are the

	L	R
U	9,9	0,8
D	8,0	7,7

TABLE 1. Stag-Hunt Game

Risk Dominant Equilibria. This can be seen from the Stag-Hunt game in Table 2, which depicts a potential game where there is a risk dominant strategy. In this game the players will gain greater utilities if they collaborate and play the combination (U,L), yet this is a risky choice since if your “opponent” fails to cooperate you receive nothing (the task cannot be completed by you alone). On the other hand if Player 1 plays D he will always gain some utility. This payoff will be less than 9 utility units which is the maximum payoff but he will earn at least 7 utility units whatever the action of Player 2 will be. The same also stands for Player 2, if he chooses to play R. For that reason the two players will be able to coordinate to the (U,L) equilibrium only if Player 1 believes that Player 2 will play L with probability greater than 0.875 and Player 2 believes that Player 1 will play U with probability greater than 0.875. This level of confidence is difficult to obtain in a distributed setting.

2.2. Fictitious Play

We have seen that distributed optimisation can be framed as the task of finding a Nash equilibrium in a game. Game-theoretical learning algorithms solve this problem by evolving each player’s initial strategy in response to the actions selected by the opponents. There is then a strong correspondence between learning in games and iterative distributed optimisation.

One of the most widely used learning techniques in game theory is fictitious play. According to fictitious play each player chooses his action according to the best response to his beliefs about the strategy of his opponent, where best response is defined as

$$BR^i(\sigma^{-i}) = \operatorname{argmax}_{s^i \in S} u^i(s^i, \sigma^{-i}) \quad (5)$$

Initially each player has some prior beliefs about the strategy that his opponent uses to choose an action. After each turn the players update their beliefs about their opponent strategy and play again the best response according to their beliefs. More formally arbitrary non-negative initial weight functions κ_0^j , $j = 1, \dots, I$ are updated using the formula:

$$\kappa_t^j(s^j) = \kappa_{t-1}^j(s^j) + I_{s_t^j=s^j} \quad (6)$$

for each j , where $I_{s_t^j=s^j} = \begin{cases} 1 & \text{if } s_t^j = s^j \\ 0 & \text{otherwise.} \end{cases}$. The mixed strategy of opponent j is estimated from the following formula:

$$\begin{aligned} \sigma_t^j(s^j) &= \frac{\kappa_t^j(s^j)}{\sum_{s^j \in S^j} \kappa_t^j(s^j)} \\ &= (1 - \frac{1}{t})\sigma_{t-1}^j(s^j) + \frac{1}{t}I_{s_t^j=s^j} \end{aligned} \quad (7)$$

This update rule is the maximum likelihood estimator for the opponents strategies if we assume that these strategies are stationary. Note therefore that an alternative to that rule is the maximum a posteriori (MAP) estimation following a Bayesian approach. The player starts with a prior belief for the set of the distributions over the actions and updates his beliefs using the Bayes rule. If the prior is chosen to be Dirichlet distributed then the two approaches are equivalent [16]. A Bayesian approach which goes beyond MAP estimation is given in [24].

Player i then chooses the action which maximises his payoffs according to his beliefs about his opponents’ play as they have been constructed using the formulas (6) and (7). These updating methods treat the environment of the game as stationary and implicitly assume that the actions of the players are sampled from a fixed probability distribution, since the recent observations have the same weight as the initial ones. This approach leads to poor adaptation when the other players choose to change their strategies.

A variation of fictitious play that treats the opponents’ strategies as dynamic and gives bigger weights to recent observations while we calculate each action’s probability is geometric fictitious play [18]. According to this variation of fictitious play we estimate each opponent’s probability to play an action s^j using the formula:

$$\sigma_t^j(s^j) = (1 - z)\sigma_{t-1}^j(s^j) + zI_{s_t^j=s^j} \quad (8)$$

where $z \in (0, 1)$ is a constant.

The main aim for a learning algorithm like fictitious play is to converge to a set of strategies that are a Nash equilibrium. For classic fictitious play (7) it has been proved [16] that if σ is a strict Nash equilibrium and it is played at time t then it will be played for all the other iterations of the game. Also that any steady state of fictitious play is a Nash equilibrium. Furthermore, even given that fictitious play is based on the incorrect assumption of a stationary environment, it has been proved that it converges for 2×2 games with generic payoffs [25], zero sum games [26], games that can be solved using iterative dominance [27] and potential games [17]. There are also games where fictitious play does not converge. An example of such a game is Shapley’s game [28].

2.3. Stochastic Fictitious Play

A player i of a game who is selecting his actions s^i using fictitious play is actually selecting his actions from his strategy space S^i according to his beliefs about his opponent’s actions. Randomisation is allowed only

in the case that the player is indifferent between his available actions, but it is very rare in generic payoff strategic form games for a player to be indifferent between the available actions [29]. Stochastic fictitious play is a modification of fictitious play that allows players to play the available actions according to a mixed strategy σ^i . This was originally introduced to allow convergence of strategies to a mixed strategy Nash equilibrium [16] but has the added advantage of introducing exploration into the process. There are two equivalent formulations of stochastic fictitious play. The first one is based on Harsanyi's purification theorem [30], which allows randomisation and chooses actions according to mixed strategies using randomly distributed perturbations of the players' estimated payoffs. An equivalent formulation [31] is for players to use a mixed strategy obtained by maximising a perturbed reward function. This allows randomisation even when the players are not indifferent about the opponents strategy. Each player i chooses his actions according to a smooth best response to the estimated strategies of the other players. The most common smooth best response to σ^{-i} is the distribution over player i 's actions which satisfies:

$$\overline{BR}(\sigma^{-i})(s^i) = \frac{\exp(u^i(s^i, \sigma^{-i})/\lambda)}{\sum_{\tilde{s}^i} \exp(u^i(\tilde{s}^i, \sigma^{-i})/\lambda)} \quad (9)$$

where λ is the randomisation parameter. Values of λ close to zero allow very little randomisation, whereas large values of λ result in complete randomisation [16].

We reinforce the fact that the difference between classic fictitious play and stochastic fictitious play is in the decision rule the players use to choose the actions. But the updating rules that are used to update the beliefs of the opponents strategies are the same in both algorithms.

3. USING PARTICLE FILTERS TO TRACK OPPONENTS' STRATEGY

3.1. Introduction

In this section we only consider inference over opponent mixed strategy in fictitious play. Separate estimates will be formed identically and independently for each opponent. We therefore consider only one opponent, drop all dependence on player i , and write s_t and σ_t for the opponent's action and strategy respectively. In fictitious play, as depicted in Figure 1, we are assuming that Player i 's opponent has a fixed mixed strategy σ over time, from which all of their actions arose. A more rational assumption about the strategies of Player i 's opponents is that they are changing over time, instead of having a fixed value. According to this assumption, at every time t Player i 's opponent has a different strategy σ_t . Figure 2 depicts fictitious play when opponent strategy is changing over time.

For the tracking scenario Player i 's objective is to predict efficiently his opponent's strategy σ_t , using

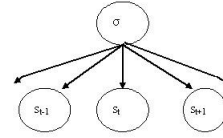


FIGURE 1. Graphical model of the standard fictitious play assumption

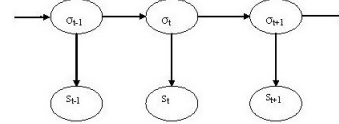


FIGURE 2. Graphical model for a more realistic fictitious play model

the information he has from the actions $s_{1:t-1}$ he has observed and his beliefs about his opponents strategies till that time $\sigma_{1:t-1}$. This objective can be represented as a Hidden Markov Model (HMM). HMMs try to predict the value of an unobserved variable x_t , the hidden state, using the observations of another variable z_t based on the following two assumptions:

- The Markovian assumption. This implies that the probability of being at any state x_t at time t depends only at the state of time $t-1$, x_{t-1} . More formally we can write this as $p(x_t|x_{t-1}, \dots, x_1) = p(x_t|x_{t-1})$.
- The conditional independence assumption. According to this assumption the observation at time t depends only on the current state x_t , and it is independent of previous observations and states, $p(z_t|x_{1:t}, z_{1:t-1}) = p(z_t|x_t)$.

If we take into account these assumptions we can write the marginal $p(x_{1:t}, z_{1:t})$ as:

$$p(x_{1:t}, z_{1:t}) = \prod_{s=1}^t p(z_s|x_s)p(x_s|x_{s-1}). \quad (10)$$

In control theory this kind of problem is referred to as the filtering problem. The process that we want to predict, the strategies of Player i 's opponents, is a non-Gaussian process and thus we use particle filters.

From the family of particle filter algorithms we will use the Sequential Importance Sampling (SIS) filter with resampling [32]. SIS is a method that performs nonlinear filtering approximation of $p(x_t|x_1, \dots, x_{t-1}, z_1, \dots, z_t)$ based on Importance Sampling integration.

To introduce importance sampling we consider the case where we want to estimate the following expectation $E_p(f(x)) = \int f(x)p(x)dx$, where $p(x)$ is the probability density of x . A natural solution of this expectation when we are able to draw M samples x^m from $p(x)$ is $E(f(x)) = \frac{1}{M} \sum_{m=1}^M f(x^m)$.

There are cases where sampling from $p(x)$ is difficult or even impossible. In these cases we can sample from a distribution $q(x)$ with heavier tails than $p(x)$ and use the Importance Sampling identity to approximate the desired integral.

$$\begin{aligned} E_p(f(x)) &= \int f(x)p(x)dx = \int \frac{p(x)}{q(x)}f(x)q(x)dx \\ &= \int w(x)f(x)q(x)dx \\ &= E_q(f(x)w(x)) \simeq \frac{1}{M} \sum_{m=1}^M f(x^m)w(x^m) \end{aligned} \quad (11)$$

where $w(x) = \frac{p(x)}{q(x)}$ and x^m are drawn from $q(x)$.

SIS with resampling, samples from the posterior of $x_{1:t}$ given the observations $z_{1:t}$. The posterior can be expressed iteratively as:

$$p(x_{1:t}|z_{1:t}) \propto p(z_t|x_t)p(x_t|x_{t-1})p(x_{1:t-1}|z_{1:t-1}). \quad (12)$$

Like importance sampling, instead of using $p(x_{1:t}|z_{1:t})$, which is infeasible to sample from, we use a distribution $q(x_{1:t}|z_{1:t})$ with similar properties. We can choose a density q which can be recursively defined as:

$$q(x_{1:t}|z_{1:t}) = q(x_t|x_{1:t-1}, z_{1:t-1})q(x_{1:t-1}|z_{1:t-1}) \quad (13)$$

Like importance sampling, we are going to use the following formula to compute the weights of the sample

$$w_t = \frac{p(x_{1:t}|z_{1:t})}{q(x_{1:t}|z_{1:t})} \quad (14)$$

substituting equations (12) and (13) to the weights equation we have:

$$w(x_{1:t}) = \frac{p(z_t|x_t)p(x_t|x_{t-1})p(x_{1:t-1}|z_{1:t-1})}{q(x_t|x_{1:t-1}, z_{1:t-1})q(x_{1:t-1}|z_{1:t-1})} \quad (15)$$

In cases where $q(x_t|x_{1:t-1}, z_{1:t}) = q(x_t|x_{t-1}, z_t)$, equation (15) becomes

$$w(x_{1:t}) = \frac{p(z_t|x_t)p(x_t|x_{t-1})p(x_{1:t-1}|z_{1:t-1})}{q(x_t|x_{t-1}, z_{t-1})q(x_{1:t-1}|z_{1:t-1})} \quad (16)$$

since $w(x_{1:t-1}) = \frac{p(x_{1:t-1}|z_{1:t-1})}{q(x_{1:t-1}|z_{1:t-1})}$ this is equivalent to

$$w(x_{1:t}) = w(x_{1:t-1}) \frac{p(z_t|x_t)p(x_t|x_{t-1})}{q(x_t|x_{t-1}, z_t)} \quad (17)$$

So we can produce a new sample and its corresponding weights, $(x_{1:t}, w_{1:t})$, using the existing particles and weights, $(x_{1:t-1}, w_{1:t-1})$, by sampling from distribution $q(x_t|x_{t-1}, z_t)$ and weight the new samples using equation (17).

Summarising, we sample M particles from a distribution $q(\cdot)$ (importance function) which has similar characteristics to the density function of the data and we associate some weights w_t^m to each sample. At the initial step we set equal weights $w_t^m = 1/M$ for all the particles because we have no observations to support other values for the weights. We sample from the importance function instead of sampling from

Basic SIS algorithm

- Draw M samples from the proposal distribution of x_t where $x_t^m \sim q(x_t^m|x_{t-1}^m, z_t)$ and $0 < m \leq M$
- Calculate the weights w_t where $w_t^m = w_{t-1}^m \cdot \frac{p(z_t|x_t^m)p(x_t^m|x_{t-1}^m)}{q(x_t^m|x_{t-1}^m)}$
- Normalise the weights
 $w_t^m = \frac{w_t^m}{\sum_{m=1}^M (w_t^m)}$
- Compute the effective sample size, N_{eff}
 $N_{eff} = \frac{1}{\sum_{m=1}^M (w_t^m)^2}$
- If $N_{eff} < N_{threshold}$ resample according to the weights using stratified resampling [33].

TABLE 2. SIS algorithm [9]

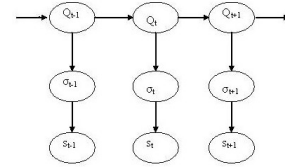


FIGURE 3. Graphical representation of the strategies propagation when we are using propensities.

the density function of the data $p(\cdot)$ because in most cases it is difficult to sample directly from p . A general algorithm for SIS filters is depicted in Table 3.

The reason for resampling is that after some iterations most of the weights will be approximately zero with very few of them large enough to influence the results. Using resampling avoids this phenomenon since it multiplies particles with high weights and discards particles with small weights in such a way as to maintain the correct importance sampling approximation.

3.2. An SIS algorithm to update fictitious play beliefs

In this section we describe how to use the particle filter within fictitious play. Since we are going to track opponent's strategy using SIS we need to sample from this probability distribution and also to propagate this sample. But we cannot propagate the strategies since we don't know the model according to which they are propagated. A way to avoid this problem is to introduce a new layer to Figure 2 with unconstrained propensities of our opponent, sample from them, and propagate this sample instead of the strategies. Figure 3 depicts the propagation of fictitious play using the propensities, where Q_t , σ_t and s_t are the propensity, the strategy and the action of Player's i opponents at time t respectively.

We assume that $Q_0 \sim MVN(0, I)$ and that Q_t depends only on its previous value Q_{t-1} according to

- Propagate M particles from the model of Q_t where $Q_t^m = Q_{t-1}^m + \eta_t^m$
- Calculate the weights w_t where $w_t^m = w_{t-1}^m \cdot p(s_t|Q_t^m) = w_{t-1}^m \cdot \frac{e^{Q_t^m(s_t)/\tau}}{\sum_{\bar{s} \in S} e^{Q_t^m(\bar{s})/\tau}}$
- Normalise the weights $w_t^m = \frac{w_t^m}{\sum_{m=1}^M (w_t^m)}$
- Compute the effective sample size, N_{eff} $N_{eff} = \frac{1}{\sum_{m=1}^M (w_t^m)^2}$
- If $N_{eff} < N_{threshold}$ resample according to the weights using stratified resampling.

TABLE 3. SIS with resampling for fictitious play algorithm

the equation:

$$Q(s_t) = Q(s_{t-1}) + \eta_t \quad (18)$$

where $\eta_t \sim MVN(0, v^2 I)$. We also assume that $\sigma_t(s) \propto \exp(Q(s)/\tau)$ for each action. We can represent the algorithm of SIS with resampling (Table 3) using the fictitious play notation, where Q_t and s_t will replace x_t and z_t respectively. So in every iteration of the particle filter we can use equation (9) to sample the M particles from our opponent's propensity, $Q_t(s_t)$, and use a form of Boltzmann distribution to update the beliefs about the opponents actions given the opponents propensities. For an action $s_t \in S$:

$$p(s_t|Q_t) = \frac{e^{(Q_t(s_t)/\tau)}}{\sum_{\bar{s} \in S} e^{(Q_t(\bar{s})/\tau)}} \quad (19)$$

where τ is a temperature constant.

We can observe in Table 3 that the weights are propagated using the following formula: $w_t = w_{t-1} \cdot \frac{p(z_t|x_t)p(x_t|x_{t-1})}{q(x_t|x_{t-1})}$. From the assumptions we have made for the distribution of Q , we know that the distribution of $Q_t|Q_{t-1}$ is a normal distribution. Since it is easy to sample from a normal distribution we can draw our M samples from the normal distribution such as $p(x_t|x_{t-1}) = q(x_t|x_{t-1})$ so we will be able to simplify the formula which is used to evaluate the weights. The simplified formula can be written $w_t = w_{t-1} \cdot p(z_t|x_t)$. Using fictitious play notation we can write $w_t = w_{t-1} \cdot p(s_t|Q_t)$. Table 4 summarises the SIS algorithm with resampling using fictitious play notation for tracking.

3.3. Experiments to test the impact of algorithm parameters

Initially simulations were employed to examine if the proposed model is able to track sufficiently a pre-specified opponent's strategy over time. Also these

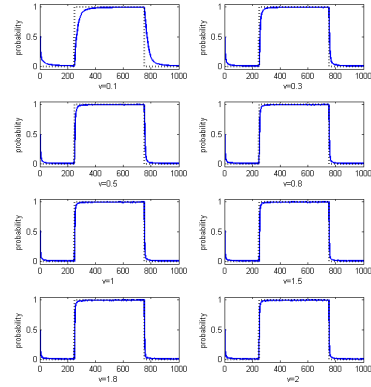


FIGURE 4. Tracking of $\sigma_t(1)$ when the true strategies are pure for different values of v . The pre-specified strategy of the opponent and its prediction are depicted as the dot and solid line respectively.

experiments were implemented to monitor the impact of the temperature parameter τ and innovation variance parameter v in the results of our algorithm. The experiments can be divided into two categories. The first category contains experiments where the opponent chooses his actions from a pure strategy space. In that category the opponent played action 2 from iteration 251 to 750 and action 1 in all the other iterations of the game. The later category allows randomisation and the opponent chooses his actions from a mixed strategy space. The opponent strategy had the following form over the t iterations of the game: $\sigma_t(1) = \frac{\cos \frac{2\pi t}{n} + 1}{2} = 1 - \sigma_t(2)$, where $n = 1000$.

Figures 4 and 5 depict the result of the tracking for different values of the standard deviation v with $\tau = 1$. In the pure strategy space we can observe that large values of v result in faster tracking as we can see in Figure 4. This happens because if we use large values of v then our algorithm actually has little memory and just follows the previous action of our opponent. This is an effective tactic in a steady environment like that of the pure strategy space. However when with large v we allowed mixed strategies then the results were too noisy and the tracking of our opponent policy was poor as we can observe in Figure 5. Note that in Figure 5 all the plots approximately track the shape of the opponents strategy, with more or less noise according to the value of v .

Figures 6 and 7 depict the tracking results of the game for different values of τ and fixed value of $v = 0.1$. Both in pure and mixed strategy space (Figures 6 and 7 respectively) we can observe that small values of τ makes the changes in the shape of the predicted distribution sharper and allow predictions close to 0 and 1. On the other hand very big values of τ i.e. $\tau = 2$ make the prediction too smooth and the prediction never reaches 1. The value of the parameter τ doesn't affect directly the memory of our system, unlike the

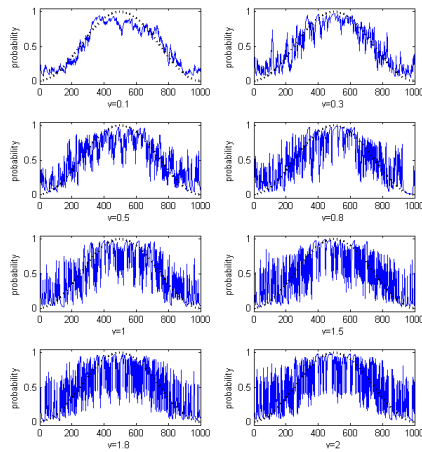


FIGURE 5. Tracking of $\sigma_t(1)$ when the true strategies are mixed for different values of v . The pre-specified strategy of the opponent and its prediction are depicted as the dot and solid line respectively.

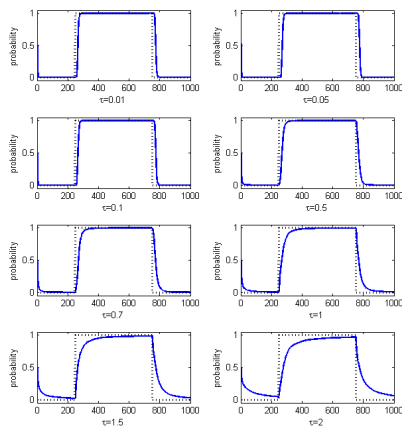


FIGURE 6. Tracking of $\sigma_t(1)$ when the true strategies are pure for different values of τ . The pre-specified strategy of the opponent and its prediction are depicted as the dot and solid line respectively.

parameter v . This can be seen from Figure 7 where all the plots have almost the same noise level.

The result of the proposed algorithm were compared with those of fictitious play when we estimate opponents strategy. Figures 8 and 9 depict the fictitious play estimate of the opponent strategy and the opponents real strategy in contrast with the predictions of the proposed algorithm. The parameters which were used for our algorithm were 0.3 and 0.7 for v and τ respectively. In both figures we can observe that the results of the proposed algorithm outperforms the results of the tracking that the fictitious play algorithm achieves.

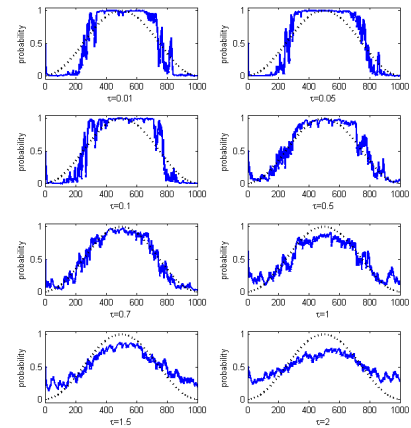


FIGURE 7. Tracking of $\sigma_t(1)$ when the true strategies are pure for different values of τ . The pre-specified strategy of the opponent and its prediction are depicted as the dot and solid line respectively.

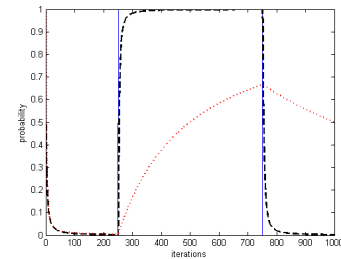


FIGURE 8. Comparing the tracking of the particle filter algorithm (dash line) with fictitious play (dot line) at a pure strategy space

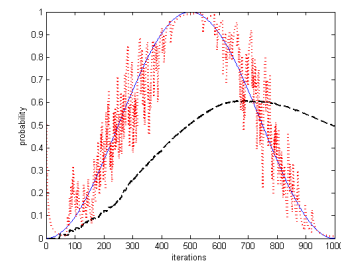


FIGURE 9. Comparing the tracking of the particle filter algorithm (dot line) with fictitious play (dash line) at a mixed strategy space

4. IMPLEMENTATION IN GAMES

4.1. Particle Filter Stochastic Fictitious Play

Until now we have managed to show that the proposed method of updating the beliefs of fictitious play performs better in tracking opponent strategy than classic fictitious play. We will combine the algorithm in Table 4 with decision making using smooth best response to introduce a new version of stochastic fictitious play. The change to the algorithm of Table

- Draw M particles from the model of Q_t
- Calculate the new σ_t
- Compute smooth best response \overline{BR}
- Choose an action according to \overline{BR}
- Observe opponent's action
- Update and normalise the weights
- Resample if necessary

TABLE 4. Stochastic fictitious play using particle filters

	Stag	Hare
Stag	8,8	0,-7
Hare	-7,0	7,7

TABLE 5. Stag hunt game

	U	M	D
U	11,11	-30,-30	0,0
M	-30,-30	7,7	6,6
D	0,0	0,0	5,5

TABLE 6. Climbing hill game with two players [34]

4 is that after propagating the Q values, we use them to predict opponent strategy and thus select an action. Specifically, we calculate the estimate

$$\sigma_t(s_t) = \sum_{m=1}^M w_t^m \cdot \frac{e^{Q_t^m(s_t)/\tau}}{\sum_{\bar{s} \in S} e^{Q_t^m(\bar{s})/\tau}} \quad (20)$$

then select an action using $\overline{BR}(\sigma_t)$ as defined in (9). A compact representation of that algorithm is depicted in Table 4.

4.2. Results

In this section we examine the performance of the particle filter stochastic fictitious play algorithm. For that purpose we compared the results of our algorithm with the ones of classic stochastic fictitious play and geometric fictitious play in three coordination games. These games are depicted in Tables 5,6 and 7. Table 5 depicts a simple coordination game with asymmetric penalties for off-diagonal play. Table 6 introduces the climbing hill game used by Claus and Boutilier [34] to demonstrate the slow convergence of algorithms that estimate action values based on the entire history of play, in which we expect geometric fictitious play and our new algorithm to significantly outperform classical stochastic fictitious play. Table 7 presents a more extreme version of this game in which three players must climb up a utility function; in this scenario each agent estimates the strategy of each opponent independently.

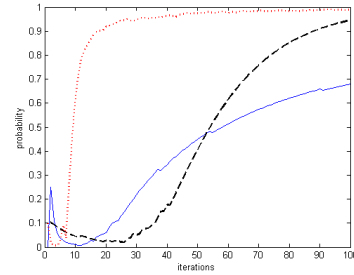


FIGURE 10. Probability of playing the (M,M) equilibrium for the particle filter algorithm (dot line), stochastic fictitious play (solid line) and geometric fictitious play (dash line) for the two player climbing hill game

We present the results of 1000 replications of a learning episode of 1000 iterations for each game. We selected the values of the parameters for each algorithm on the premise that the algorithms with these parameters have the best results in the tracking experiment with pre-specified opponents strategy. The values of the parameters for the particle filter tracking algorithm were 0.3 and 0.7 for v and τ respectively. The value of the learning parameter z of geometric fictitious play was set 0.05. To allow the same exploration for all algorithms we set the randomisation parameter λ in the smooth best response function equal to 1 for all the algorithms. For the Stag hunt game the particle filter algorithm converged to the Pareto efficient equilibrium (Stag, Stag) in 92% of the replications. For each replication of 1000 iterations we computed the mean payoff. After the end of the 1000 replications the overall mean of the 1000 payoff means was computed. The overall mean payoffs for player 1 and 2 were 7.78 and 7.74 respectively. The classic stochastic fictitious play converged to the (Stag, Stag) equilibrium in all replications but very slowly with overall mean payoff 7.53 for both players. The results for geometric fictitious play were better than our algorithm since it converged always to the Pareto efficient equilibrium with mean payoff 7.9 for both players.

In the Climbing hill game with two players our algorithm failed to reach the equilibrium (U,U) where the players gain the highest utility and it converged to the (M,M) equilibrium. The overall mean payoffs for player 1 and 2 were 6.98. In this game the stochastic and geometric fictitious play also converged to the (M,M) equilibrium. The overall mean payoffs for player 1 and 2 were 6.95 and 6.96 respectively. The results are very close because both algorithms converge quickly to the equilibrium point. If we take a game with 100 iterations then the results are not so close. The mean payoff for stochastic fictitious play is 6.46, for geometric fictitious play 6.53 and for the proposed algorithm is 6.85. The difference we observe is because our algorithm converges faster to the equilibrium than the other two algorithms. We can observe this difference in Figure 10.

	U	M	D	U	M	D	U	M	D
U	0	0	0	-300	70	80	100	-300	90
M	0	50	40	-300	60	0	0	0	0
D	0	0	30	0	0	0	0	0	0
	U			M			D		

TABLE 7. Climbing hill game with three players. Player 1 selects rows, Player 2 selects columns, and Player 3 selects the matrix.

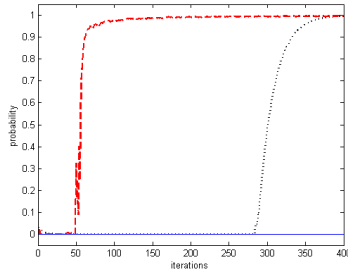


FIGURE 11. Probability of playing the (U,U,D) equilibrium for the particle filter algorithm (dash line), stochastic fictitious play (solid line) and geometric fictitious play (dot line) for the three player climbing hill game

In the more complicated environment of the Climbing hill game with three players our algorithm also outperformed both geometric and stochastic fictitious play. The overall mean payoffs for player 1,2 and 3 were 98.6 when payoffs for dynamic and stochastic fictitious play were 91.7 and 70.3 respectively. Stochastic fictitious play didn't converge to the Nash equilibrium after 1000 replications. Also when we are concerned about the speed of convergence of the particle filter algorithm and geometric fictitious play our algorithm outperforms geometric fictitious play. This can be seen if we reduce the iterations of the game to 200. Then the overall mean payoff of our algorithm is 93.2 utility units when for geometric fictitious play is 63.12. This is because our algorithm needs 50 iterations to reach the Nash equilibrium when geometric fictitious play needs at least 300. This difference is depicted in Figure 11.

5. VEHICLE TARGET ASSIGNMENT GAME

We compared the results of our algorithm against those of geometric fictitious play in the vehicle target assignment game that is described in [21]. Agents here should coordinate to achieve a common goal which is to maximise the total value of the targets that are destroyed. In particular in a specific area we place I vehicles and J targets. For each i , $S^i = J$, i.e the actions are simply the choice of target to engage. Each vehicle can choose only one target to engage but a target can be engaged by many vehicles. The probability that player i has to destroy a target j is p_{ij} if it chooses to engage j . We assume that the probability each agent

has to destroy a target is independent from the other agents so the probability a target j is destroyed from the vehicles that engage it is $1 - \prod_{i:s^i=j} (1 - p_{ij})$. Each of the targets have a different value V_j . The expected utility that is produced from the target j is the product of its value V_j and the probability it has to be destroyed by the vehicles that engage it. More formally we can express the utility that is produced from target j as:

$$U_j(s) = V_j \left(1 - \prod_{i:s^i=j} (1 - p_{ij}) \right) \quad (21)$$

The global utility is then the sum of the utilities of each target.

$$U_{global}(s) = \sum_j U_j(s) \quad (22)$$

5.1. Results

Two different instances of the vehicle target assignment game were used to compare the two algorithms. In the first there are twenty vehicles with twenty targets placed uniformly at random in a unit square. A vehicle i has probability to destroy a target j proportional to the inverse of its distance from this target $1/d_{ij}$. In this example there is no range restriction, so that all the targets can be engaged by all the vehicles and there is also interaction between all the vehicles. In the second example we placed fifty targets and vehicles in a unit square. The probability to destroy a target is the same as the former example but here each vehicle is able to interact only with other vehicles that are at most 0.6 distance units away, and can only engage targets that are at most 0.3 distance units away. We compared our algorithm against geometric fictitious play in both examples.

In both cases the positions of the vehicles and the targets were the same for predictive and geometric fictitious play, as well as the values of the targets. The vehicles had to "negotiate" with the other vehicles (players) for a fixed number of negotiation steps before they choose a target to engage.

A negotiation step begins with each player choosing a target to engage and it ends by the agents exchanging this information with the others, and updating their beliefs about their opponents' strategies based on this information. The targets that the players will decide to engage will be the ones of the joint action of the final negotiation step for both learning algorithms.

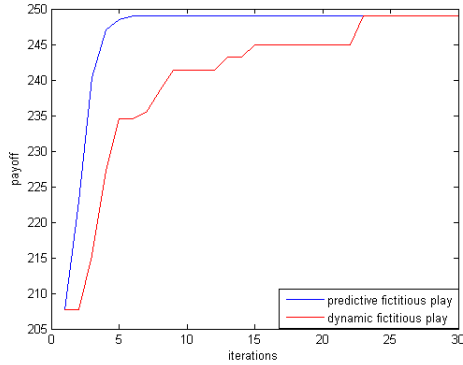


FIGURE 12. Utility of particle filters FP and dynamic FP for the vehicle target assignment game

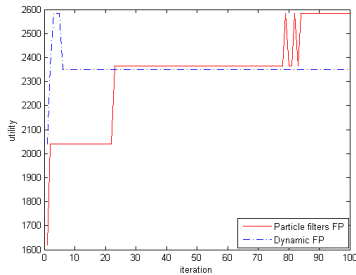


FIGURE 13. Utility of particle filters FP and dynamic FP for the range restricted vehicle target assignment game

For the range restricted version of the game each vehicle could communicate, exchange information and update his beliefs only with the players that were in its communication range of 0.6 distance units.

Figures 12 and 13 depict the results that we obtain for particle filter fictitious play and geometric fictitious play in the full range and the restricted range version of the vehicle target assignment game respectively. The values of the parameters for the particle filter tracking algorithm were 0.3 and 0.7 for v and τ respectively in the full range version of the game. In the range restricted version smoother estimators of the opponents' strategies were needed and for that reason we change the parameter τ to $\tau = 10$. These values are different from the ones we propose in section 3 because the likelihood of opponents strategies are very sharp in this case, and thus if we use the values we proposed previously we eventually react only to the last observed action. So to obtain smooth estimations of opponents strategies we had to choose a significantly larger value of τ than 0.7 which was the proposed value in section 3. The value of the learning parameter z of geometric fictitious play was set to 0.1. Like in the strategic form games for both algorithms we set the randomisation parameter λ in the smooth best response function equal

to 1, allowing the same randomisation. In Figure 12 we observe that both algorithms result in the same final reward but the particle filter algorithm reaches that reward faster than the geometric fictitious play. Our algorithm converges to a higher reward in the range restricted version of the game since the geometric fictitious play converged to a joint action with smaller reward as is depicted in Figure 13

6. DISASTER MANAGEMENT SCENARIO

We also have tested our algorithm in a disaster management scenario as described in [1]. Consider the case where a natural disaster has happened (an earthquake for example) and because of this N_I simultaneous incidents occurred in different areas of a town. In each incident j a different number of people $N_p(j)$ were injured. The town has a specific number of ambulances N_{amb} available that are able to collect the injured people. An ambulance i can be at the area of the incident j in time T_{ij} and has capacity c_i . We will assume that the total capacity of the ambulances is larger than the number of injured people. Our aim is to allocate the ambulances to the incidents in such a way that the average time $\frac{1}{N_{amb}} \sum_{i:s^i=j} T_{ij}$ that the ambulances need to reach the incidences is minimised while all the people that are engaged in the incident will be saved. Then we can formulate this scenario as follows. There are N_{amb} players that can each choose one of the N_I incidents as actions. The utility to the system of an allocation is:

$$u(s) = -\frac{1}{N_{amb}} \sum_{j=1}^{N_I} \sum_{i:s^i=j} T_{ij} - \sum_{j=1}^{N_I} \min(0, N_p(j) - \sum_{i:s^i=j} c_i) \tag{23}$$

where $i = 1, \dots, N_{amb}$ and s^i is the action of player i . The emergency units had to “negotiate” with each other and choose the incident that will be allocated using fictitious play.

The first component of the utility function expresses our first aim, to allocate the ambulance to an incident as fast as possible. Thus the agents have to choose an incident with small T_{ij} . The second objective, which is to save all the injured people that are engaged in an incident, is expressed as the second component of the utility function. It is a penalty factor that adds to the average time the number of the people that were not able to be saved. Like the vehicle target assignment game each player can choose only one incident to help, but in each incident more than one player can provide his help.

6.1. Results

We tested our algorithm in simulations with 3 and 5 incidents, and 10, 15 and 20 available ambulances. We run 200 trials for each of the combinations of ambulances and incidents. Since this scenario is

		%complete	% saved	f_{fp}/f_{opt}
3 incidents	10 ambulances	94.0	99.69	1.1852
	15 ambulances	96.0	99.84	1.0942
	20 ambulances	96.5	99.89	1.0858
5 incidents	10 ambulances	86.5	96.26	1.4729
	15 ambulances	87.0	99.45	1.3754
	20 ambulances	84.5	99.5	1.2898

TABLE 8. Results after 200 negotiation steps for the three performance measures.

NP-complete [1] our aim is not to find the optimal solution after 200 iterations, but to reach a sufficient or a near optimal solution. The algorithm presented here is “any-time”, since the system utility generally increases as the time goes on, and therefore interruption before termination results in good, if not optimal actions. In each of the 200 instances the time T_{ij} that an ambulance needs to reach an incident was a random number uniformly distributed between zero and one, the capacity of each ambulance was an integer uniformly distributed between one and four, and the total number of injured people involved in each incident was a uniformly distributed integer between $\frac{c_t}{2 * N_I}$, $\frac{c_t}{N_I}$, where c_t is the total capacity of the emergency units $c_t = \sum_{i=1}^{N_{amb}} c_i$.

The results we obtain are the average of 200 different instances for each combination of number of incidents and number of ambulances. In each trial we allowed 200 negotiation steps. The randomisation parameter λ was set to be 0.001. This results in a decision rule which approximates best response. Furthermore parameter v was set to 0.3 for all the simulation scenarios. Also we set the temperature parameter of the Boltzman equation τ to be 10 because smoother predictions of the opponents strategies are needed when a lot of opponents influence the utility, as in the range restricted vehicle target assignment game.

We tested our algorithm using the same performance measures as [1]. We compared the solution of our algorithm against the centralised solution which can be obtained using binary integer programming. In particular we compared the solution of our algorithm against the one we obtained using Matlab’s command *bintprog* using the ratio $\frac{f_{fp}}{f_{opt}}$, where f_{fp} is the negative of the utility that the agents could gain if they used the particle filter fictitious play and f_{opt} is the utility that the agents should gain if they were using the solution of *bintprog*. We used the negative utility of the particle filters fictitious play because *bintprog* solves the minimisation form of the problem so the solutions of the two algorithms had opposite sign. Furthermore we measured the percentage of the instances in which all the casualties were rescued, and the overall percentage of people that were rescued.

Table 8 includes the results we obtain using the results of the last step of negotiations between the ambulances for the disaster management scenario. We

can observe that the results we obtain for 3 incidents cases are better, in two of the three performance measurements we used, than the cases where 5 incidents had occurred. The percentage of the people that finally the emergency units were able to collect was similar for cases of 3 and 5 incidents. A reason that we observe this phenomenon is that in the 5 incidents cases, because of the problem’s structure, more people are engaged and also the ambulances have more allocation choices. These facts result in a more complicated problem than the one where 3 incidents occurred, so it is easier for the algorithm to be trapped into a local minimum.

The differences we can observe from Matlab’s centralised solution can be explained from the structure of the utility function. The first component of the utility is a number between zero and one since it is the average of the times, T_{ij} , that the ambulances need to reach the incidents. On the other hand the penalty factor, even in the cases where only one person is not collected from the incidents, is greater than the first component of the utility. Thus a local search algorithm like particle filter fictitious play initially searches for an allocation that collects all the injured people, so the penalty component of the utility will be zero, and afterwards for the allocation that minimises also the average time that the ambulances needed to reach the incidents.

When we consider the two performance measures that take into account only the number of people that have been collected, then we can notice that the results we obtain within the two groups of simulations, with 3 and 5 incidents respectively, are similar for all the different combinations of the available ambulances we examined. But this is not the case when we take into consideration the ratio of f_{fp}/f_{opt} , where we always had better results when the number of the available ambulances was increased. Especially in the simulations where 5 incidents had occurred the results were better when 20 ambulances were available when we compared with the occasions where 10 and 15 ambulances were available, by 13.69% and 6.34% respectively.

This difference can be explained by the choice of parameters λ , and τ . When the utility of a player is influenced by his predictions over many opponents strategies then these predictions should be smooth so he will not respond only to the last action of one of his opponents, that may be an exploratory action. Thus we have chosen a big value for the smoothing parameter τ . This value of τ is suitable for a large number of opponents, like the case of the 20 ambulances, but it is not working efficiently in less complicated scenarios like the one of the 10 ambulances. Also, because of the small chance the agents had to explore after setting parameter $\lambda = 0.01$, the agents could coordinate more easily, but pay the cost that when they were trapped in a local minimum they weren’t able to escape.

In addition the particle filter fictitious play algorithm performed better than the RNN presented in [1], when

		Iterations			
		50	100	150	200
3 incidents	10 ambulances	49.0	75.5	90.0	94.0
	15 ambulances	60.5	83.5	92.0	96.0
	20 ambulances	56.0	76.0	86.5	96.5
5 incidents	10 ambulances	12.5	39.5	69.5	86.5
	15 ambulances	13.0	36.5	67.5	87.0
	20 ambulances	21.0	42.0	64.0	84.5

TABLE 9. Percentage of solutions in which the capacity of the ambulance in every incident was enough to cover all injured people for different stopping times of the negotiations, 50, 100, 150 and 200 iterations of the Fictitious Play algorithm.

we consider the percentage of the cases in which all the injured people were collected and the overall percentage of people that were rescued. The percentage of instances where the proposed allocations by the RNN could collect all the casualties were from 25 to 69 percent while the respective results of particle filter fictitious play algorithm were from 84.5 to 96.5. Also the overall percentage of people that were rescued from the RNN algorithm were between 85 and 98.5 percent, when our algorithm manages to collect, in all cases, more than 99 percent of the casualties. As expected from a centralised solution the results of the ratio of the utility that produced by the RNN by the utility produced by *bintprog* is better since the maximum value of the ratio was 1.1. Another reason that the results of our algorithm were considerably different from these of RNN in this score is that we included all the cases when in [1] only the cases where all the casualties were rescued were included in the evaluation of the ratio.

We also examined how the results are influenced by the number of iterations we use in each of the 200 trials of the game. For that reason we compared all the performance measures of our algorithm for 50,100,150 and 200 negotiation steps. Tables 9, 10, and 11 show the results that we would have obtained if in each instance of the simulations we had stopped the negotiations between the emergency units after 50, 100, 150 and 200 iterations, for the % of cases that all the injured people were collected, the average % of people that were collected and the average value of the ratio between the solution of our algorithm and the optimal solution of *bintprog* respectively. In addition the boxplots of Figures 14 and 15 depict the changes in the results we obtained after 50, 100, 150 and 200 iterations for the total percentage of people that were saved and the ratio f_{fp}/f_{opt} respectively for the case with the 3 incidents and the 20 ambulances.

Table 9 shows that the more negotiations steps we allow to the emergency units the greater the chances are to collect all the casualties. Thus we need the 200 iterations and a lot of communication resources to achieve that goal. As expected this affected also the results of the ratio f_{fp}/f_{opt} through the negotiation

		Iterations			
		50	100	150	200
3 incidents	10 ambulances	94.76	98.37	99.42	99.69
	15 ambulances	96.16	99.15	99.63	99.84
	20 ambulances	96.78	98.80	99.52	99.89
5 incidents	10 ambulances	86.51	95.04	98.21	99.26
	15 ambulances	90.68	96.14	98.61	99.45
	20 ambulances	92.75	96.99	98.74	99.50

TABLE 10. Average percentage of injured people collected for different stopping times of the negotiations, 50, 100, 150 and 200 iterations of the Fictitious Play algorithm.

		Iterations			
		50	100	150	200
3 incidents	10 ambulances	1.5253	1.2917	1.2070	1.1852
	15 ambulances	1.3619	1.1448	1.1072	1.0942
	20 ambulances	1.3019	1.1606	1.1163	1.0858
5 incidents	10 ambulances	2.5304	1.7918	1.5740	1.4729
	15 ambulances	2.1472	1.6321	1.4495	1.3754
	20 ambulances	1.9373	1.5308	1.3730	1.2898

TABLE 11. Average percentage of the ratio f_{fp}/f_{opt} for different stopping times of the negotiations, 50, 100, 150 and 200 iterations of the Fictitious Play algorithm.

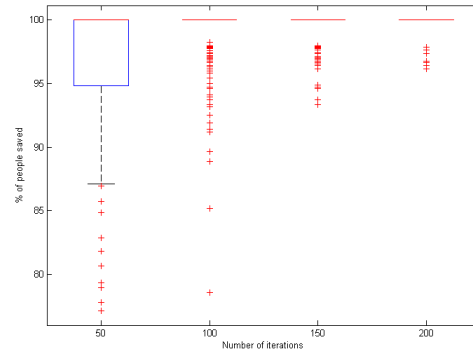


FIGURE 14. Percentage of injured people collected for different stopping times of the negotiations, 50, 100, 150 and 200 iterations of the Fictitious Play algorithm for the case with 3 incidents and 20 ambulances.

steps. As it is presented in Table 11 we can see that the ratio drops as the negotiation steps are increasing for all the cases, especially in the 5 incidents cases where the differences in the percentage of the instances in which all casualties were rescued was higher as the negotiation steps increased. We can also observe the influence of the percentage of the instances all casualties collected in the ratio f_{fp}/f_{opt} in Figure 15. The median were always less than 1.2 but at the early stages of the negotiations the values were more spread and also more outliers (depicted as +) existed. On the other hand, as depicted in Table 10, if we mainly focus on the total number of people that the ambulances can collect then even with less negotiation (and so less communication cost) a large percentage of the casualties can be collected. In

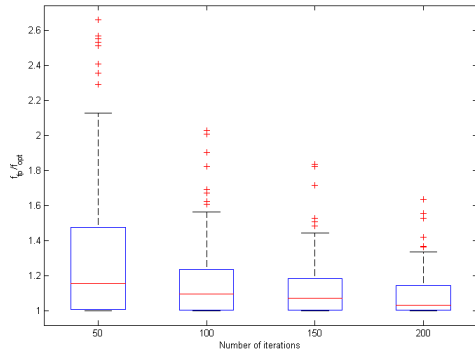


FIGURE 15. Ratio f_{fp}/f_{opt} for different stopping times of the negotiations, 50, 100, 150 and 200 iterations of the Fictitious Play algorithm for the case with 3 incidents and 20 ambulances.

particular the difference in the results after 100 and 200 negotiation steps were between 2.88% and 4.28%. We can observe also in Figure 14 that the difference from 50 to the 200 iterations is that as we increase the iterations fewer outliers (depicted as +) exist.

7. CONCLUSIONS

Distributed optimisation problems can be formulated as a game. It is therefore natural to try to find optima by employing game-theoretical learning algorithms. The classic learning algorithm is fictitious play, but this is formed on an (incorrect) stationarity assumption. Therefore we have introduced a modification of fictitious play that uses Hidden Markov Models for the beliefs about opponent strategies.

We have examined the impact that the two parameters τ and v have in particle filter algorithm results. Parameter v controls the memory of our algorithm. The larger v is the less memory our algorithm has. For values of v greater than 1.5 our algorithm mimics our opponent's previous action. This can be derived in the way we propagate our beliefs about our opponent's strategy: $Q_t = Q_{t-1} + \eta_t$, where $Q_0 \sim N(0, I)$ and $\eta_t \sim N(0, v^2 I)$. Introducing a noise greater than the scale of the Q values means that the values of Q_{t-1} lose their importance and the algorithm chooses all the Q_t values to give large $\sigma_t(s_t)$.

The other parameter is the temperature τ of the Boltzman equation. This parameter has greatest impact on the range and the sharpness our estimation of the opponent's strategy will have. Our prediction of the opponent's strategy is computed based on the following formula:
$$\frac{e^{Q_t(s_t)/\tau}}{\sum_{\bar{s} \in A} e^{Q_t(\bar{s}_t)/\tau}}.$$

The value of τ affects the value that the exponential will have and so the amplitude and the shape of our prediction. Large values of τ make the differences between the probabilities of the actions smaller than

they are, and small values of τ have exactly the opposite result. In Section 3.3 we show that a value of $\tau = 0.7$ produces predictions that are neither too smooth nor too sharp. Nevertheless there are cases where the players should choose smoother predictions of the opponents strategies than the real ones. An example of such an occasion is when there are many opponents and the likelihood function has a very sharp peak. Then even small changes in opponents strategies have a high impact on the actions of the other players. In these cases the players should choose a reasonably large value of τ , higher than 0.7.

Finally, the last parameter of the algorithm λ has an effect on the randomisation of stochastic fictitious play. Large values of λ allows randomisation even if the player is not indifferent between the actions [16]. The player can choose an action which is not the best response to his beliefs with bigger probability. When this parameter tends to zero there is no randomisation.

Stochastic fictitious play with particle filters to update its beliefs about the opponents strategy performs better in the three strategic form games than classic fictitious play. Although it was outperformed by geometric fictitious play in the potential game, the speed of convergence to the equilibrium point in both variations of the climbing hill game is faster when using particle filters. Furthermore particle filter fictitious play converged faster to the same solution as geometric fictitious play in the full range vehicle target assignment game, and also converge to a better solution for the restricted range version of this game. In the disaster management scenario we compared the solution of particle filter fictitious play with that of Matlab's *bintprog*. Our algorithm performed worse than *bintprog*, but this can be explained by the structure of the utility function and the local searching nature of the particle filter fictitious play algorithm that tended to choose solutions that were focused on collecting all the injured people. Furthermore, because of the big value of the smoothing parameter and the decision rule that approximated best response when the algorithm was trapped in a local minimum, it was difficult to continue exploring. The pre-planning algorithm of [1] performed better when we concern as performance measure the ratio of each algorithm's utility by the one of *bintprog*'s. But our algorithm performed better when we concerned the number of cases that all the casualties were rescued, and the overall number of people that the emergency units were able to rescue.

8. ACKNOWLEDGMENTS

The authors gratefully acknowledge BAE SYSTEMS and ESPSRC funding for the ALADDIN project (EP/C548051/1)

REFERENCES

- [1] Gelenbe, E. and Timotheou, S. (2008) Random neural networks with synchronized interactions. *Neural Computing*, **20**(9), 2308–2324.
- [2] National Fire Protection Association (NFPA) (2007), Standard on Disaster/Emergency Management and Business Continuity Programs. NFPA, 2007.
- [3] Kitano, H., Todokoro, S., Noda, I., Matsubara, H., Takahashi, T., Shinjou, A. and Shimada, S. (1999) Robocup rescue: Search and rescue in large-scale disaster as a domain for autonomous agents research. In *IEEE International Conference on Systems, Man, and Cybernetics (SMC '99)*, **6**, 739–743.
- [4] Kho, J., Rogers, A. and Jennings, N. R. (2009) Decentralise control of adaptive sampling in wireless sensor networks. *ACM Transactions on Sensor Networks*, **5** (3).
- [5] van Leeuwen, P. (2002) Scheduling aircraft using constraint satisfaction. In *Electronic Notes in Theoretical Computer Science*. Elsevier, 252–268.
- [6] Stranjak, A., Dutta, P.S., Rogers, A. and Vytelingum, P.V. (2008) A multi-agent simulation system for prediction and scheduling of aero engine overhaul. In *Proceedings of the 7th International Conference on Autonomous Agents and Multiagent Systems (AAMAS '08)*
- [7] Bertsekas, D. (1982) Distributed dynamic programming. *IEEE Transactions on Automatic Control*, **27**(3), 610–616.
- [8] Silva, C.A., Sousa, J.M.C., Runkler, T.A. and Sa da Costa, J.M.G. (2009) Distributed supply chain management using ant colony optimization. *European Journal of Operational Research*, **199**(2), 349–358.
- [9] Petcu, A., and Faltings, B. (2005) A scalable method for multiagent constraint optimization. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence, IJCAI-05*, 266–271.
- [10] Modi, P., Shen, W., Tambe, M. and Yokoo, M. (2004) ADOPT: Asynchronous Distributed Constraint Optimization with Quality Guarantees. *Artificial Intelligence*, **161**, 149–180.
- [11] Mailler, R. and Lesser, V. (2006) Asynchronous Partial Overlay: A New Algorithm for Solving Distributed Constraint Satisfaction Problems. *Journal of Artificial Intelligence*, **25**, 529–576.
- [12] Aji, S.M. and Mc Eliece, R.J. (2000) The generalised distributed law. *IEEE transactions on Information Theory*, **46**, 325–343.
- [13] Hart, S. and Mas-Colell, A. (2000) A simple adaptive procedure leading to correlated equilibrium. *Econometrica*, **68**(5), 1127–1150.
- [14] Yokoo, M. and Hirayama, K. (1996) Distributed Breakout Algorithm for Solving Distributed Constraint Satisfaction Problems. *Second International Conference on Multiagent Systems (ICMAS-96)*, 401–408
- [15] Brown, G.W. (1951) Iterative Solutions of Games by Fictitious Play. In *Activity Analysis of Production and Allocation*, T.C. Koopmans (Ed.). New York: Wiley.
- [16] Fudenberg, D. and Levine, D. (1998) *The theory of Learning in Games*. The MIT Press
- [17] Monderer, D. and Shapley, L. (1996) Potential Games. *Games and Economic Behavior*, **14**, 124–143.
- [18] Benaim, M., Hofbauer, J. and Hopkins, Ed. (2008) Learning in games with unstable equilibria. *Journal of Economic Theory*, **144**(4), 1694–1709.
- [19] Fudenberg, D. and Tirole, J. (1991) *Game Theory*. The MIT Press
- [20] Wolpert, D. and Tumer, K. (1999) An overview of collective intelligence. In J. M. Bradshaw, editor, *Handbook of Agent Technology*. AAAI Press/MIT Press.
- [21] Arslan, G., Marden, J. and Shamma, J. (2007) Autonomous Vehicle-Target Assignment: A Game Theoretical Formulation. *Journal of Dynamic Systems, Measurement, and Control*, **129**, 584–596.
- [22] Chapman, A., Rogers, A. and Jennings, N. R. (2008) A Parameterisation of Algorithms for Distributed Constraint Optimisation via Potential Games. *Tenth International Workshop on Distributed Constraint Reasoning (DCR '08)*, 99–113.
- [23] Nash, J. (1950) Equilibrium Points in n-Person Games *Proceedings of the National Academy of Science, USA*, **36**, 48–49.
- [24] Rezek, I., Leslie, D. S., Reece, S. and Roberts, S. J., Rogers, A., Dash, R. K. and Jennings, N.R. (2008) On Similarities between Inference in Game Theory and Machine Learning. *Journal of Artificial Intelligence Research*, **33**, 259–283.
- [25] Miyasawa, K. (1961) On the convergence of learning process in a 2x2 non-zero-sum two person game. *Research Memorandum No 33, Princeton University*.
- [26] Robinson, J. (1951) An iterative Method of solving a game. *Annals of Mathematics*, **54**, 269–301.
- [27] Nachbar, J. (1990) “Evolutionary” selection dynamics in games: Convergence and limit properties. *International Journal of Game Theory*, **19**, 58–89.
- [28] Shapley, L. (1964) *In advances in Game theory*. Princeton, Princeton University
- [29] Fudenberg, D. and Kreps, D. (1993) Learning Mixed Equilibria. *Games and Economic Behavior*, **5**, 320–367
- [30] Harsanyi, J.C. (1973) Games with randomly distributed payoffs: A new rationale for mixed-strategy equilibrium points. *International Journal of Game Theory*, **2**, 1–23.
- [31] Hofbauer, J. and Sandholm, W. H. (2002) On the Global Convergence of Stochastic Fictitious Play. *Econometrica*, **70**(6), 2265–2294
- [32] Gordon, N. J., Salmond, D. J. and Smith, A.F.M. (1993) Novel approach to nonlinear/non-Gaussian Bayesian state estimation. In *IEEE Proceedings in Radar, Sonar and Navigation*, **140**(2), 107–113.
- [33] Arulampalam, S., Maskell, S., Gordon, N. and Clapp, T. (2002) A Tutorial on Particle Filters for Online Nonlinear/Non-Gaussian Bayesian Tracking. *IEEE Transactions on Signal Processing*, **50**(2), 174–188
- [34] Claus, C. and Boutilier, C. (1998) The Dynamics of Reinforcement Learning in Cooperative Multiagent Systems. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence*.