Benchmarking Hybrid Algorithms for Distributed Constraint Optimisation Games

Archie C. Chapman, Alex Rogers and Nicholas R. Jennings
Electronics and Computer Science
University of Southampton
Southampton SO17 1BJ, UK
{ac05r,acr,nrj}@ecs.soton.ac.uk

ABSTRACT

In this paper, we consider algorithms for distributed constraint optimisation problems (DCOPs). Using a potential game characterisation of DCOPs, we decompose six distributed DCOP algorithms, taken from the game theory and computer science literatures, into their salient components. We use these components to construct five novel hybrid algorithms. We then empirical evaluate all eleven DCOP algorithms in a series of graph colouring experiments. Our experimental results show the existence of several performance trade–offs, which may be exploited by a system designer to tailor a DCOP algorithm to suit their mix of requirements.

Categories and Subject Descriptors

G.1.6 [**Optimization**]: Global optimization

General Terms

Distributed constraint optimisation, potential games

1. INTRODUCTION

Large–scale systems are difficult to solve optimally, often because communication restrictions make it difficult, costly or impossible to collect all the necessary information at the location where a solution is to be computed. These difficulties then motivate the use of distributed methods of optimisation. Given this background, much attention has been focussed on using machine learning techniques to solve or approximate solutions to large–scale optimisation problems (e.g. [3]). Within this context, in this paper, we focus on distributed constraint optimisation problems (DCOPs). Specifically, we are interested in algorithms in which the actors are distributed and can only communicate with their peers. For thus reason, we exclude centralised approaches in which all of the information needed to solve a problem is accessible to a single decision maker. We also exclude distributed algorithms that rely on highly structured interactions, such as algorithms that run on a pre-computed tree (e.g. DPOP [16]), because such interactions often become prohibitively costly as the size of the problems increases. We call the remaining algorithms *completely distributed algorithms*.

Against this background, we use a unified analytical framework based on potential games to decompose six of the major completely distributed algorithms for DCOPs. We use the framework to construct a number of novel learning algorithm hybrids. The potential game framework allows us to construct these algorithms in a principled manner. We then compare their performance to that of the existing DCOP algorithms in a series of simulation experiments in a graph colouring domain. By doing so, we identify the effects that using various components have on the behaviour of an algorithm.

The paper progresses as follows. In the next section we introduce our potential game characterisation of DCOPs. We begin Section 3 by describing our algorithm decomposition, and then fit six existing algorithms to this framework, before constructing our novel hybrid algorithms. In Section 4 we discuss the results of a series of graph colouring experiments. Section 5 concludes.

2. DCOPS AS POTENTIAL GAMES

This section begins with an overview of noncooperative games, before focusing on potential games in particular. We then introduce constraint optimisation problems, and show how a DCOP may be expressed as a potential game. This result is used in subsequent sections, to construct a parameterisation of the design space for DCOP algorithms, and to analyse their behaviour and explore the algorithm design space.

2.1 Potential Games

A noncooperative game, $\langle N, \{S_i, u_i\}_{i\in N}\rangle$, is comprised of a set of agents $N = 1,\ldots,n$, and for each agent $i \in N$, a set of *strategies* S_i, with $\cup_{i=1}^{N} S_i = S$, and a *utility function* $u_i : S \to \mathbb{R}$. A joint strategy profile $s \in S$ is referred to as an *outcome* of the game, where S is the set of all possible outcomes, and each agent's utility function specifies the payoff they receive for an outcome by the condition that, if and only if the agent prefers outcome s to outcome s', then $u_i(s) > u_i(s')$. We will often use the notation $s = \{s_i, s_{-i}\}$, where s_{-i} is the complimentary set of s_i.

An agent's goal is to maximise its own payoff, conditional on the choices of its opponents. Stable points in such a system are characterised by the set of *Nash equilibria*, which are defined as a joint strategy profile, s^*, such that no individual agent has an incentive to change to a different strategy:

$$u_i(s_i^*, s_{-i}^*) - u_i(s_i, s_{-i}^*) \geq 0 \quad \forall\, s_i,\ \forall\, i. \tag{1}$$

The class of finite *potential games* are used to describe many problems in multi-agent systems, in particular congestion problems on networks [18], and more recently, power control, channel selection

and scheduling problems in wireless networks [10, 11], target assignment problems [2] and job scheduling [23].

DEFINITION 1. *A function $P : S \rightarrow \mathbb{R}$ is a* potential *for a game if $\forall\, i \in N$:*

$$P(s_i, s_{-i}) - P(s'_i, s_{-i}) = u_i(s_i, s_{-i}) - u_i(s'_i, s_{-i}) \quad \forall\, s_i,\ s'_i \in S_i. \quad (2)$$

A game is called a potential game if it admits a potential [15].

Intuitively, a potential is a function of action profiles such that the difference induced by a unilateral deviation equals the change in the deviator's payoff. The existence of a potential function implies a strict joint preference ordering over game outcomes. This, in turn, ensures that the game possesses two desirable properties.

First, every finite potential game possesses at least one pure–strategy equilibrium [18]. To see this, let P be a potential for a game. Then s is an equilibrium point for the potential game if and only if for every $i \in N$,

$$P(s) \ \geq\ P(s'_i, s_{-i}) \quad \forall\, s'_i \in S_i.$$

Consequently, if P admits a maximal value in S (true by definition for finite S), then the game possesses a pure–strategy Nash equilibrium. Now, pure–strategy equilibrium are particularly desirable in decentralised agent–based systems, as they imply a stable, unique outcome. Mixed strategy equilibria, on the other hand, imply a probabilistically stable, but stochastically variable strategy profile.

Second, every potential has the *finite improvement property* [15]. An *improvement step* in a game is a change in one player's strategy such that its utility is improved. A *path* is a sequence of steps, $\phi = (s^0, s^1, s^2 \ldots)$, in which exactly one player changes their strategy at each step, and ϕ is an *improvement path* in a game if for all t, $u_i(s^{t-1}) < u_i(s^t)$ for the deviating player i at step t. A game is said to have the finite improvement property if every improvement path is finite. Now, in a potential game, for every improvement path $\phi = (s^0, s^1, s^2, \ldots)$ we have, by Equation 2:

$$P(s^0) \ <\ P(s^1) \ <\ P(s^2) \ <\ \ldots$$

Then, as S is a finite set, the sequence ϕ must be finite, so every potential has the finite improvement property. The finite improvement property ensures that the behaviour of agents who independently play 'better-responses' in each period of the repeated game converges to a Nash equilibrium. Taken together, the two properties discussed above ensure that many simple adaptive processes converge to a pure–strategy Nash equilibrium in potential games.

2.2 DCOP Games

We now consider DCOPs, and show that a game-theoretic formulation of a DCOP is a potential game. This is important as it allows us to apply the convergence results presented above to many other algorithms, which, in turn, will allow us to structure our parameterisation of the DCOP algorithm design space in a principled manner.

A constraint optimisation problem is formally represented by a set of variables $V = \{v_1, v_m, \ldots\}$, each of which may take one of a finite number of states or values, $s_j \in S_j$, a set of constraints $C = \{c_1, c_2, \ldots\}$, and a global utility function, u_g, that specifies preferences over configurations of states of variables in the system. A constraint $c = \langle V_c, R_c \rangle$ is defined over a set of variables $V_c \subset V$ and a relation between those variables, R_c, which is a subset of the Cartesian product of the domains of each variable involved in the constraint, $\prod_{v_j \in V_c} S_j$. A function that specifies the reward for satisfying, or penalty for violating, a constraint is written $u_{c_k}(s_{c_k})$, where s_{c_k} is the configuration of states of the variables V_{c_k}. Using this, the global utility function aggregating the utilities from satisfying or violating constraints commonly takes the form:

$$u_g(s) = \sum_{c_k \in C} u_{c_k}(s).$$

This aggregation is strictly monotonic, in that an increase in the number of satisfied constraints results in an increase in the global utility. Constraints may be ascribed different levels of importance by simply weighting the rewards for satisfying them, or by using a positive monotonic transform of constraint reward [19]. The objective is then to find a global configuration of states, s^*, such that:

$$s^* \in \ \underset{s \in S}{\operatorname{argmax}} \ \ u_g(s).$$

Given this, a DCOP is produced when a set of autonomous agents each independently control the state of a subset of the variables, but share the goal of maximising the rewards for satisfying constraints. A DCOP game is a simple formulation that explicitly model the strategic dependencies between the variables each agent controls. Without loss of generality, we consider the case where each agent controls only one variable. As such, we can use the terms 'state of a variable' and 'strategy of an agent' interchangeably, and will notate the set of agents involved in a constraint by N_c. Finally, the set of constraints that i is involved in is notated by C_i.

A DCOP game is formulated by assigning each agent a *private utility function*, $u_i(s)$, which is dependent on both its own state and the state of other agents in the system. There is some flexibility in the choice of utility function, however, it is vital that an agent's utility only increases when the global solution quality is improved. This is done by setting each agent's utility equal to its local effect on the global utility function, which, in a DCOP, is given by the sum of the payoffs for constraints that agent i is involved in:

$$u_i(s) = \sum_{c_k \in C_i} u_{c_k}(s_i, s_{\nu(i)}).$$

Now, each agent desires to maximise its private utility, and agents are allowed to adjust their strategies in repeated plays of the game. Distributed solutions to the DCOP are produced by the independent actions of the agents in the system. Consequently, these solutions are located at the Nash equilibria of the DCOP game.

We now state the key result we have derived, and upon which the rest of the paper hinges, placing DCOP games in the class of potential games.

THEOREM 1. *Every strictly monotonic DCOP game in which the agents' private utilities are given by their local effects on the global utility function is a potential game.*

PROOF. Since a change in i's strategy only affects the neighbours of i, $\nu(i)$, the following statements hold:

$$\begin{aligned} u_i(s_i, s_{\nu(i)}) - u_i(s'_i, s_{\nu(i)}) &= \sum_{c_k \in C_i} u_{c_k}(s_i, s_{\nu(i)}) - \sum_{c_k \in C_i} u_{c_k}(s'_i, s_{\nu(i)}) \\ &= \sum_{c_k \in C} u_{c_k}(s_i, s_{-i}) - \sum_{c_k \in C} u_{c_k}(s'_i, s_{-i}) \\ &= u_g(s_i, s_{-i}) - u_g(s'_i, s_{-i}), \end{aligned}$$

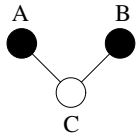
where the third line flows from the second by definition. □

Thus, u_g is an exact potential function for a DCOP game where the agents' private utilities are given by their local effects on the value of the global utility function. Consequently, any change in state that

increases an agent's private utility also increases the global utility of the system.

In the case of binary constraints, the game played between pairs of agents sharing a constraint can be easily expressed in matrix form. One widely studied binary constraint optimisation problem is graph colouring, and in Section 4 we run a series benchmarking experiments sing graph colouring as our problem. For these reason, we present an example graph colouring game.

EXAMPLE 1. *In graph colouring, neighbouring nodes share constraints, which are satisfied if the nodes are in differing states. Consider the following graph colouring problem, where each node can be either black or white, and the associated constraint game:*



$u_c(s_i, s_j) =$

	B	W
B	(0, 0)	(1, 1)
W	(1, 1)	(0, 0)

In this example, agents A and B each effectively play the game above with agent C, while agent C plays the composite game below, constructed by combining the constraint games it plays with each neighbour. In the table below, A and B are column players and C is the row player, payoffs are (u_A, u_B, u_C).

s_A, s_B		B, B	B, W	W, B	W, W
s_C	B	(0, 0, 0)	(0, 1, 1)	(1, 0, 1)	(1, 1, 2)
	W	(1, 1, 2)	(1, 0, 1)	(0, 1, 1)	(0, 0, 0)

Finally, a potential function for the game is given below:

s_A, s_B		B, B	B, W	W, B	W, W
s_C	B	0	1	1	2
	W	2	1	1	0

Now that we have shown that DCOP games are potential games, we are assured that at least one pure-strategy Nash equilibrium exists. Furthermore, to prove that the globally optimal joint profile corresponds to a Nash equilibrium, assume that the optimal point is not a Nash equilibrium. Then there must be some agent that can alter its state to improve its utility, which in turn will improve the global utility, which contradicts the assumption that the optimal point is not a Nash equilibrium. Despite that, note that in most cases many Nash equilibria exist, some of which will be sub-optimal.

In the next section we describe the processes by which agents adjust their states in order to arrive at an equilibrium. However, before continuing, we make one general comment regarding both the interpretation of the repeated game and the strategy adaptation process. We assume that agents suffer from extreme myopia, and do not look beyond the immediate rewards for taking an action, so the only Nash equilibria that are supported are the Nash equilibria of the one-shot DCOP game.

3. DISTRIBUTED DCOP ALGORITHMS

A comparison of completely distributed algorithms for DCOPs is worthwhile in itself, however, we are interested furthering the state of the art by exploiting our potential game characterisation of DCOPs. To this end, we begin this section by introducing a unified framework that captures the major completely distributed DCOP algorithms. We then examine six existing algorithms from the game theory and computer science literature, and fit them to our framework. These algorithms are: the distributed stochastic algorithm (DSA) [6]; the maximum–gain messaging algorithm (MGM) [21, 13]; fictitious play (FP) [5, 17]; regret matching (RM) [9]; spatial adaptive play (SAP) [22]; and smooth fictitious play (smFP) [7, 8]. Finally, we construct five hybrid algorithms, using components taken from the six existing algorithms. These hybrid algorithms are then evaluated alongside the six existing algorithms in the experiments described in Section 4. Pseudo code for the algorithms is given in Appendix A.

3.1 DCOP Algorithm Framework

The framework which use breaks a DCOP algorithm into three components. To begin with, given an appropriate trigger, the individual agents follow the same basic two–stage routine. First is the *state evaluation*. Each algorithm has a *target function* that it uses to evaluate its prospective states. The target functions are typically functions of payoffs, and sometimes take parameters that are set exogenously to the system or updated online. Second is a *decision* on which action to take, based on the preceding state evaluations. The *decision rule* refers to the procedure by which an agent uses its evaluation of states to decide upon an action to take. The third component is the system–wide process that controls which agent adjusts its state at each point in time, given by an *adjustment schedule*. In many algorithms (particularly those addressed in the game theory literature), the scheduling mechanism is often left unspecified, or is implicitly random. However, some algorithms are identified by their use of specific adjustment schedules that allow for preferential adjustment or parallel execution. Furthermore, in some cases the adjustment schedule is embedded in the decision stage of the algorithm. Note that communication does not figure explicitly in this framework. Information is communicated between agents for two purposes: (i) to calculate the value of their target function, or (ii) to run the adjustment schedule. Thus, the communication requirements of each algorithm depend on these two stages.

3.2 Existing DCOP Algorithms

Using the framework described above, we now decompose six major DCOP algorithms into their constituent components, and examine how these components affect the algorithms' properties when they are used in repeated potential games.

3.2.1 Distributed Stochastic Algorithm

DSA uses the immediate payoff for selecting a state, $u_i(s_i, s_{-i})$ as its target function, and for a decision rule, uses the *argmax* function (i.e., selects the state with the greatest payoff). As such, this algorithm is effectively a greedy, local search algorithm. As an adjustment schedule, it employs a *random parallel* schedule, in which each agent has some probability p, known as the degree of parallel executions, of actually changing their state at any time step [23]. This adjustment schedule is motivated by observing that when neighbouring agents adapt their states at the same time, they can inadvertently move their joint state to a globally inferior outcome, a phenomenon known as 'thrashing'. The random parallel schedule cannot ensure that no thrashing takes place, but by selecting an appropriate value of p, or decreasing p along an appropriate schedule, thrashing behaviour can be minimised. Additionally, the finite improvement property ensures that DSA almost surely converges to a Nash equilibrium in repeated potential games.

3.2.2 Maximum–Gain Messaging Algorithm

Similar to DSA, MGM also uses the immediate payoff and *argmax* function as a target function and decision rule, respectively. The algorithms differ purely by the adjustment schedule they employ: MGM uses a schedule that give preference to agents that can achieve the greatest gains, which we call *maximum–gain* schedule. This involves agents exchanging messages regarding the maximum gain

they can achieve. If an agent can achieve the greatest gain out of all its neighbours, then it implements that change, otherwise it maintains its current state. The maximum–gain messaging adjustment schedule avoids all thrashing, as no two neighbouring agents will ever move at the same time. MGM converges to a Nash equilibrium, and furthermore, is an anytime algorithm.

3.2.3 Fictitious Play

The term 'fictitious play' is often used to denote a family of adaptive processes which use the expected payoff over historical frequencies of actions as a target function. Let i's belief over its opponents' joint strategy profiles, $q_i^t(s_{-i})$, be given by the frequency of times it observes each joint profile. Each agent's expected payoff given this belief, FP_i^t, is then:

$$FP_i^t = \sum_{S_{-i}} q_i^t(s_{-i}) u_i(s_i, s_{-i}).$$

This target function can also be specified recursively, which only requires agents to maintain a measure of the expected payoff for each state, rather than the full action history:

$$FP_i^t = 1/t \left[u_i(s_i, s_{-i}^t) + (t-1) FP_i^{t-1} \right],$$

where $u_i(s_i, s_{-i}^t)$ is the fictitious payoff to i for each element of S_i given its opponents' profile at t. In this subsection, we consider the main version, referred to as FP, which uses the *argmax* decision rule in conjunction with the 'flood' adjustment schedule, under which all agents adjust their state simultaneously. Now, all versions of fictitious play that use historical frequencies as a target function and the *argmax* decision rule (regardless of the adjustment schedule used) have the property that if play converges to a pure strategy profile, it must be a Nash equilibrium, because if it were not some agent would eventually change their strategy. Additionally, strict Nash equilibria are absorbing; if a strict Nash equilibrium is played once it is played from then on.

Specifically regarding FP, in repeated potential games, this algorithm *converges in beliefs*; that is, each agents estimate of its opponents' strategies, which is used to calculate each of its own strategy's expected payoff, converges as time progresses [14]. Consequently, each agent's best response strategy also converge to a Nash equilibrium profile. This process induces some stability in an agent's choice of strategy, minimises thrashing and cycling behaviour, and generally speeds up convergence.

3.2.4 Regret Matching

Another approach that can be used to speed up convergence is to measure the average 'regret' for not taking an action, written AR_i^t, where regret is the difference in payoff for choosing a state and the state that was actually chosen at a particular time:

$$AR_i^t = 1/t \sum_{\tau=1}^{t} [u_i(s_i, s_{-i}^\tau) - u_i(s_i^\tau, s_{-i}^\tau)]. \quad (3)$$

This target function can be specified recursively, only requiring the agents to maintain a measure of average regret for each state:

$$AR_i^t = 1/t \left[u_i(s_i, s_{-i}^t) - u_i(s^t) + (t-1) AR_i^{t-1} \right].$$

This target function is used by [9] to construct the RM algorithm. RM then takes these regret values and maps them through a linear probabilistic decision rule to produce a mixed strategy with probabilities in direct proportion to the target value of each state, with negative regrets given zero probability:

$$P_{s_i} = \frac{u_i(s_i, s_{-i}^t)}{\sum_{s_i \in S_i} u_i(s_i, s_{-i}^t)}.$$

Finally, agents choose not to change their state with a small probability proportional to the size of their strategy space and scaled by the difference between their highest and lowest possible payoffs. Thus, the algorithm effectively uses a random parallel schedule, the same schedule as DSA. RM converges to the set of correlated equilibria (a generalisation of Nash equilibria) in all finite games, however, it does not necessarily converge to Nash equilibria.

3.2.5 Spatial Adaptive Play

Like DSA and MGM, SAP uses the immediate payoff for selecting a state as its target function. However, it differs in both its decision rule and adjustment schedule. SAP uses the multinomial logit decision rule [1], also known as the Boltzmann distribution:

$$P_{s_i}(\eta) = \frac{e^{\eta^{-1} u_i(s_i, s_{-i}^t)}}{\sum_{s_i \in S_i} e^{\eta^{-1} u_i(s_i, s_{-i}^t)}}. \quad (4)$$

Here states are chosen in proportion to their reward, but their relative probability is controlled by η, a temperature parameter. If $\eta = 0$ then the *argmax* function results, while $\eta = \infty$ produces a uniform distribution across strategies, which results in the state of the system following a random walk. The choice between the *argmax* decision rule and a probabilistic decision rule serves an important purpose. In the former case, the algorithm converges quickly, and may even be anytime, but it may not be able to escape from the basin of attraction of a local maximum. The latter case adds ergodicity to the algorithm, which allows it to escape from sub-optimal, local maxima, but at the cost of sometimes degrading the solution quality. Depending on the specifics of the problem at hand, the temperature can be kept constant or may be decreased over time according to some annealing schedule. The later case is referred to in the online reinforcement learning literature as a 'greedy in the limit with infinite exploration' (GLIE) decision rule [20].

SAP also uses a *sequential random* adjustment schedule, which randomly gives one agent at a time the opportunity to adjust its strategy, with agents selected by some probabilistic process. The motivation for using this adjustment schedule is grounded in the convergence proofs for many of the adaptive procedures taken from the game theory literature. In particular, the finite improvement property of potential games directly implies that agents who play a sequence of 'better responses' converges to a Nash equilibrium in a finite number of steps. This property, in conjunction with results regarding the convergence of GLIE processes,[1] is used to prove the convergence of SAP to the global optimum of the potential function in repeated potential games [22, Chapter 6].

3.2.6 Smooth Fictitious Play

smFP uses the expected payoff over historical frequencies of actions as a target function and the flood adjustment schedule, and so differs from FP only in its use of a probabilistic decision rule. Typically, smFP uses the multinomial logit decision rule, which results in a version of smFP that is known to converge to the set of Nash equilibria in potential games [12].

3.3 Hybrid DCOP Algorithms

As noted in the introduction, one of our motivations for developing a parameterisation of the DCOP algorithm design space is an interest in exploiting the potential synergies that exist by combining components of different existing algorithms. In this section, we investigate five such hybrid algorithms constructed from the components identified in the previous subsection, and guided by the

[1] For example, the appropriate annealing schedule for the logistic decision rule is $\eta \propto 1/\log t$.

properties of potential games. These particular hybrids were selected because, firstly, we expect them to display the types synergies we are interested in, and secondly, because they show that the behaviour of an algorithm can be predicted by identifying which components from our parameterisation are used in the algorithm.

Specifically, the first algorithm, greedy spatial adaptive play, removes the stochasticity in state choice from SAP. The second and third hybrid algorithms are spatial fictitious play and distributed stochastic fictitious play, both which augment standard fictitious play with a new adjustment schedule. Fourth, smooth spatial fictitious play adds stochasticity to the newly–described spatial fictitious play algorithm. Finally, the fifth, maximum–gain message regret matching, uses maximum gain priority to schedule the agents' adjustment process. We note that other combinations of algorithm components and parameter settings are possible, and indeed were examined. However, here we report only these five as they are the most interesting, the best performing, or give the clearest examples of how altering a component of the algorithm affects its behaviour.

3.3.1 Greedy Spatial Adaptive Play

In greedy spatial adaptive play (gSAP), the multinomial logit decision rule of SAP is replaced by the *argmax* function, with the target function and random sequential adjustment schedule of SAP retained. This substitution is motivated by the observation that, although standard SAP converges to an optimal solution, it converges very slowly [2], and that by removing the stochasticity in state decisions, the algorithm will converge more quickly.

Of course, by using the *argmax* decision rule, the global optimality of the long run (i.e. as $t \to \infty$) solution produced by gSAP is lost, but by the finite improvement property, the algorithm is still guaranteed to converge to a strict Nash equilibrium in potential games. Consequently, in practice, we expect gSAP to produce relatively good quality solutions in better time than SAP.

3.3.2 Spatial Fictitious Play

Spatial fictitious play (SFP) is a combination of FP with SAP, in which the asynchronous move adjustment schedule of SAP replaces the simultaneous moves (flood schedule) of standard FP, while retaining the expected payoff over historical frequencies as the target function and *argmax* as the decision rule. Such an algorithm has been suggested in theory and has been shown to converge in potential games [4], but has never been tested empirically.

Like gSAP and MGM, the convergence of this form of fictitious play is a corollary of the finite improvement property of ordinal potential games. In more detail, under SFP, if player i switches from s_i to s_i', then there exists an improvement path from (s, s_{-i}) to (s_i', s_{-i}), so any sequence of switches traverses an improvement path and consequently converges to a Nash equilibrium.

Besides its proven convergence properties, another motivation for considering this algorithm is that a system in which agents move asynchronously is closer to reality in most computational systems. In particular, in distributed systems, the assumption of synchronous moves, as in the flood schedule, is somewhat dubious, so SFP is a better fit to many practical situations.

3.3.3 Distributed Stochastic Fictitious Play

In distributed stochastic fictitious play (DSFP), agents use a random parallel schedule rather than a flood schedule as in FP. Like SFP, expected payoff over historical frequencies as the target function and *argmax* as the decision rule are retained. Additionally, the motivation for using such an algorithm in distributed systems is the same as that of SFP.

Now, although DSFP has not been proven to converge, the general results for all fictitious play-like algorithms — stationary strategy profiles are necessarily Nash equilibria and strict Nash equilibria are absorbing [8] — hold, so we expected to see the algorithm converge. Just as importantly, we predict that the effects of using different adjustment schedules are independent of the target function used, a hypothesis which we can test by comparing DSFP to FP and SFP, in conjunction with a comparison of DSA and gSAP.

3.3.4 Smooth Spatial Fictitious Play

Like SFP, agents using smooth spatial fictitious play (smSFP) follow a random sequential schedule rather than a flood schedule. Additionally, as in smFP, the logit decision rule is used, and expected payoff over historical frequencies is retained as the target function. Again, adding stochasticity should result in better quality solutions at a cost in terms of an increase in the average convergence time. Furthermore, by considering smSFP, we can test our prediction that the effects of using a probabilistic decision rule are independent of the target function used.

3.3.5 Maximum–Gain Message Regret Matching

In the maximum–gain message regret matching algorithm (MGM–RM), agents use the maximum gain priority adjustment schedule. We consider this combination because we believe that the maximum–gain messaging schedule will improve the speed of convergence of the RM algorithm, as when used in an algorithm that also uses immediate rewards as a target function, the algorithm produces better solutions at each iteration.

4. EXPERIMENTS

we wish to test how the different algorithm components affect the solutions generated by an algorithm. To this end, we benchmark the algorithms discussed in the preceding section in a series of distributed graph colouring problems. Graph colouring, as well as being a standard problem addressed in the literature, can be broadly applied as a formalism for modelling many important problems in multi–agent systems, such as scheduling problems, channel selection in wireless communication networks, and multi–agent resource allocation problems.

4.1 Experimental Design

The set of graph colouring problems we consider is made up of 60 random 80 node graphs. The number of states (colours) available to the agents is varied so that the problems may be successfully satisfied, with 20 graphs colourable in three colours, another 20 colourable in four colours and 20 colourable in five colours. The mean connectivity (the average number of links per node) of each graph is held constant at three for all the graphs considered.[2] In order to produce statistically significant results, 50 runs of each graph where completed by each algorithm. A run consisted of a maximum of 80 cycles, which was seen to be adequate for most of the algorithms to converge to a steady state.

The performance of each algorithm is measured by five metrics, chosen so that we may identify the effects of the various algorithm components. Three metrics are used to measure the optimality of each algorithm. Firstly, the ratio of the average utility of the solutions produced to the optimal, across all time steps, is used to

[2]Preliminary experiments found that altering the mean connectivity of the graph had no effect on the quality of the solutions produced.

broadly compare the overall performance of algorithms. Secondly, the average of the ratio of the utility of the solution at the termination of the run to the the optimal solution, is used to identify those algorithms that may be slow to converge but, in the end, provide high quality solutions. Thirdly, the proportion of runs in which the algorithm finds an optimal solution is used to isolate those algorithms that frequently produce an optimal solution. The speed of convergence of each algorithm is summarised in the average time it takes the algorithm to find a Nash equilibrium. We use Nash equilibrium because, as predicted by the theory, we expect to see the algorithms converging to sub–optimal Nash equilibria, and we are interested in measuring the rate of convergence to a solution, not the rate of convergence to an optimal solution. And finally, the communication requirement of each algorithm is measured by the average number of messages communicated per node per cycle. Additionally, to clearly show the behaviour induced by the different algorithms, we plot the average utility against time.

4.2 Results and Discussion

The simulation results are shown in Table 1, with 95% confidence intervals in parentheses. Where they are useful for illustrating interesting behaviour, the relevant time series of the simulations are presented in Figure 1. We begin our discussion of the results by considering the six existing DCOP algorithms discussed in Section 3.2: DSA, SAP, MGM, FP, RM, and smFP. We then relate these results to those of the five novel hybrid algorithms discussed in Section 3.3: gSAP, SFP, DSFP, smSFP and MGM–RM.

Regarding the existing algorithms, consider first their solution quality. As predicted by the theory, RM frequently does not converge to a Nash equilibrium (it is only guaranteed to converge to the set of correlated equilibria) and rarely finds the optimal solution. However, the average utility and utility of the final solution of the RM algorithm are both very good. Beyond that, in absolute terms, all of the remaining algorithms perform well. This is to be expected, as all have proven convergence properties. In particular, SAP outperforms all the existing algorithms. This advantage is particularly noticeable in its ability to find the optimal solution or a solution very close to the optimal one by the termination of the simulated run, as measured by the ratio of the solution's utility at the termination to the the optimal utility and the proportion of runs in which it finds an optimum. The reason for this is based in part on its guaranteed asymptotic convergence to the global optimum, but as importantly, on the fact that the sequential adjustment schedule ensures no thrashing occurs. The remaining algorithms — DSA(0.4), MGM, FP and smFP– perform roughly equivalently by all three optimality criteria. Worthy of particular mention is the performance of DSA(0.4). We chose $p = 0.4$ as a good representative of the spectrum of DSA algorithms as it provided very good solution quality in a timely manner at a low communication cost. A marginal increase in p only very slightly improved the speed of convergence, with little or no improvement in the solution quality, and, consistent with the results seen in [23], values of $p < 0.85$ saw a rapid deterioration in the solution quality and convergence speed of the algorithm. Values of $p < 0.4$ saw a drop off in the speed of convergence with no improvement in solution quality.

Second, we considered the speed of convergence of the existing algorithms. The algorithms using expected reward over historical frequencies of play as their target function, FP and smFP, both converge very quickly, as can be seen in Figure 1. This result matches well with the motivation for using fictitious play–like algorithms, and is an effect of convergence in the belief state used by these algorithms. By comparing these two algorithms, we can also draw an inference about the effect of adding stochasticity to decision choice. Specifically, the smFP algorithm is constructed by replacing the *argmax* function of FP with the multinomial logit decision rule. This induces stochasticity, which should permit the algorithm to move from low–payoff equilibria (or local maximum of the potential function) to higher–payoff equilibria, or ideally a global maximum. The cost of this behaviour is an increase in the time taken for a stochastic algorithm to converge. These predictions are indeed borne out in the observed behaviour of smFP compared to FP. Adding stochasticity to the decision increases the proportion of runs that produce an optimal solution by the termination of the simulation by around 18%, while the average length of time to convergence to Nash equilibrium increasing by around 40%. This trade–off is seen in Figure 1, where FP outperforms smFP for the first 18 time steps. After 18 time steps, FP had almost always converged, whereas at the same point, smFP's solutions were still improving, and continue to improve for another 10 or 20 time steps, giving a better quality solution at the termination of the simulation. These results indicate that the choice over whether to use a stochastic decision rule depends on the user's preference for improved solution quality over increased search time. Regarding the remaining algorithms, on average, both DSA and SAP converge relatively quickly. This is a consequence of the adjustment schedules they use: DSA converges quickly because it allows parallel execution of state adjustments, while SAP converges quickly because it minimises thrashing and cycling behaviour. On the other hand, the relatively slow convergence of MGM is also a result of its adjustment schedule. RM does not necessarily converge to Nash equilibria, so, unsurprisingly, has long convergence times.

Third, consider the communication resources used by the algorithms. By this metric RM is the superior algorithm: RM comes within 6% of the solution quality of SAP at $1/16^{th}$ of the communication cost. Furthermore, both FP and smFP also use relatively few messages, and taken together with RM, we conclude that the averaging mechanisms employed by these algorithms act to stabilise their choice of state, which reduces their communication use. At the other end of the scale, the MGM algorithm is particularly poor, due to the fact that the maximum–gain messaging schedule is a two-stage adjustment process that requires at least one full round of messages to be sent between neighbouring agents.

From looking at the performance of the existing algorithms, we can make the following comments regarding the effects of using various algorithm components. Firstly, when using immediate payoffs as a target function and the *argmax* decision rule, the random parallel adjustment schedule produces lower convergence times and better quality solutions than other algorithms using immediate payoffs as a target function, because it limits the effects of thrashing while allowing for a reasonable level of parallelisation. Secondly, FP and smFP converge quickly, and rely on few messages. Thirdly, RM, although it does not converge to Nash equilibrium, produces good quality solutions using remarkably few messages. Fourthly, using a stochastic decision rule improves the final solution quality, but at a cost of increased convergence time and increased communication costs. Finally (and related to the previous comment), the optimal, but slow, convergence of SAP is evident when its unremarkable average utility is compared to the high–quality final solutions and high proportion of optimal configurations it produces.

We now consider the novel hybrid algorithms, and relate their performance to that of the existing algorithms. Our aims in doing so

Table 1: Graph Colouring Results

Algorithm	Average Utility	Ratio to Optimum at T	Proportion Optimal at T	Avg Time to Converge	Avg Comm.
DSA(0.4)	0.968 (0.001)	0.984 (0.001)	0.19 (0.01)	19.3 (0.3)	0.47 (0.01)
MGM	0.960 (0.002)	0.971 (0.002)	0.42 (0.02)	31.0 (1.0)	5.57 (0.05)
FP	0.964 (0.001)	0.972 (0.001)	0.25 (0.02)	8.7 (0.2)	0.32 (0.01)
RM	0.929 (0.001)	0.956 (0.001)	0.02 (0.01)	74.8 (0.4)	0.05 (0.00)
SAP	0.973 (0.001)	0.992 (0.000)	0.37 (0.02)	22.2 (0.3)	1.32 (0.03)
smFP	0.963 (0.001)	0.975 (0.001)	0.30 (0.02)	12.4 (0.2)	0.52 (0.01)
gSAP	0.983 (0.001)	0.990 (0.001)	0.69 (0.02)	5.3 (0.1)	0.94 (0.03)
SFP	0.966 (0.001)	0.971 (0.001)	0.24 (0.02)	7.9 (0.2)	0.45 (0.02)
DSFP(0.4)	0.958 (0.001)	0.969 (0.001)	0.21 (0.01)	19.2 (0.5)	0.19 (0.01)
smSFP	0.966 (0.001)	0.974 (0.001)	0.27 (0.02)	11.6 (0.3)	0.52 (0.02)
MGM–RM	0.931 (0.001)	0.955 (0.001)	0.03 (0.01)	70.0 (0.6)	3.97 (0.02)

are to test if the specific results above have more general applicability, and to see if the predictions we made in Section 3.3 about the performance of the novel algorithms are true.

First, regarding gSAP, by substituting the *argmax* rule for the logistic decision rule used in SAP, the convergence time of the algorithm is substantially reduced, but at a cost in terms of decreased solution quality, as expected. However, as shown in Figure 1, the trade–off is quite one–sided, with the average solution produced by SAP only overtaking gSAP's solution near the end of the simulation. That said, longer simulations showed that the solutions produced SAP continued to improve, while the average solution quality produced by gSAP remained at about the level seen in Table 1. This difference is expected, because gSAP is prone to becoming stuck in sub–optimal Nash equilibria, whereas in theory SAP will converge to the global optimum. Also, the number of messages communicated by gSAP is less than the number for SAP, which is a result of its improved convergence time.

Second, SFP produces solutions of the same quality as the standard, synchronous moves FP, and converges slightly more quickly, but with a $1/3$ increase in the number of messages communicated. Using asynchronous moves does nothing to alter the speed of convergence of the beliefs used in these algorithms' target function, however, in the early stages of the run, it does act to reduce any thrashing (as gSAP does in comparison to DSA) that may occur before beliefs have begun to converge. Overall, this is a positive result, because no loss of performance implies that FP or SFP may be substituted for each other in real multi–agent systems, depending on the system designer's ability to synchronise the actions of the constituent agents.

Third, DSFP also produces solutions that are comparable in quality to FP, nut the use of a random parallel schedule does slow down the convergence of the algorithm and reduces its ability to find an optimal configuration of states. That said, it produces solutions using considerably fewer messages than either FP or DSA, because, agents are given fewer opportunities to change their state under the random parallel schedule than under the flood schedule, converging beliefs in the expected payoff over historical frequencies of actions target function act to reduce thrashing and cycling behaviour. These results are indicative of a synergy in combining the DSA and FP algorithms. Furthermore, in comparison to RM, another algorithm that requires very few messages to run, DSFP(0.4) is $5^1/_2$ times more likely to find an optimal configuration.

Fourth, smSFP exhibits an increase in the proportion of runs in which it finds the optimal solution by the termination of the simulation is offset by an increase in the time it takes for the algorithm to converge, when compared to SFP. This is a result of using a probabilistic decision rule, and corroborates with the comparisons of gSAP to SAP and FP to smFP. The quality of the solutions produced by smSFP is comparable to FP, smFP and SFP. Additionally, in comparison to smFP, smSFP converges more quickly. This is more evidence for the ability of the sequential random schedule employed by SAP, gSAP, SFP and smSFP to speed up the convergence of an algorithm.

Fifth, using the maximum gain messaging schedule did little to improve the quality of the solutions produced by using average regret as a target function and a linear decision rule. This is because the states suggested by the target function and decision rule do not necessarily improve the solution quality, causing the algorithm to 'stick' in configurations that are not Nash equilibria.

When taken together, these results indicate that gSAP and SAP are the best performing algorithms evaluated here. However, the magnitude of their superiority over some of the other algorithms may be exacerbated by our choice of metric. In particular, compare gSAP to DSA (the same comparison can be made between, on one side, SAP, FP, smFP, SFP and smSFP, and on the other side, MGM, DSFP and MGM–RM). In 80 cycles, gSAP provides $80 \times n$ individual adjustment opportunities to agents in the system. DSA, on the other hand, only provides $p \times 80 \times n$ adjustment opportunities (in the case of $p = 0.4$, $32 \times n$). This skews some of the metrics in the favour of gSAP. If we adjust the average ratio of the utility to the optimum results by comparing average solution quality of gSAP up to $t = 32$ to the average solution quality of DSA(0.4) at $t = 80$, we find values of 0.972 and 0.968, respectively. Similarly, to compare the average convergence time of gSAP and DSA(0.4), we can multiply the result for DSA by p, giving a scaled average convergence time of $0.4 \times 19.3 = 7.72$. While in general these adjusted results do not alter the ranking of the algorithms, they do indicate that the differences in performance between the algorithms are not as large as indicated by the raw figures.

In summary, the main effects of using different parameterisation components are as follows. First, the random parallel adjustment schedule limits the effects of thrashing, however this trait is only useful in algorithms that use immediate payoffs as a target function, as other target functions use averaging techniques (i.e. beliefs or average regrets) to eliminate thrashing and cycling. Second, the sequential random adjustment schedule always improves the convergence time of an algorithm, but for target functions other than

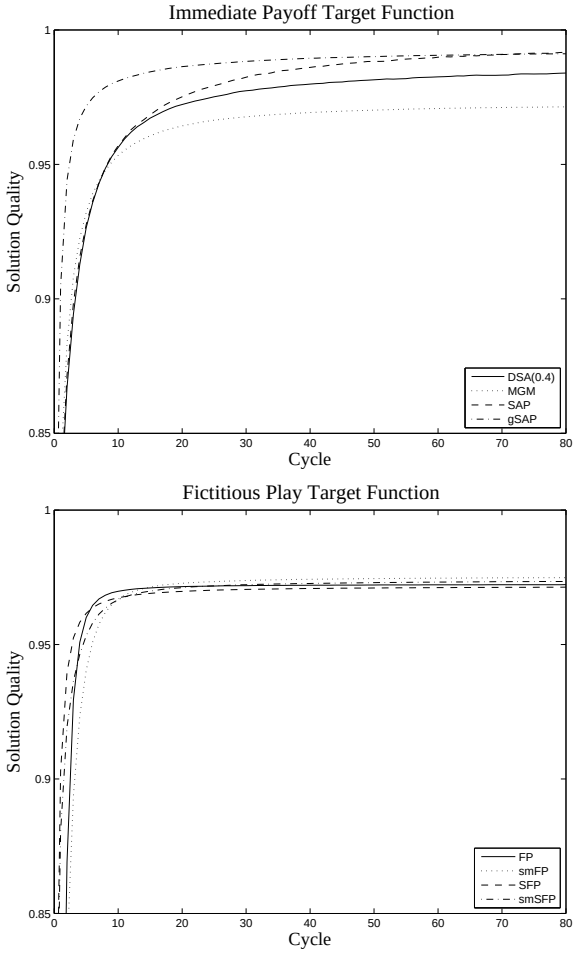


Figure 1: Average Utility vs. Time

immediate payoffs, it does not improve solution quality. Third, due to the averaging mechanism they employ, algorithms that use the expected payoff over historical frequencies of actions as a target function converge quickly and rely on few messages, because few time–steps are wasted in thrashing and cycling behaviours. Fourth, RM produces good quality solutions using remarkably few messages, again due to the averaging mechanism it employs, but does not converge, and furthermore, its performance cannot be improved by employing alternative adjustment schedules. Finally, using a stochastic decision rule improves the final solution quality, but at a cost of increased convergence time and increased communication costs.

5. CONCLUSION

In this paper, we focus on completely distributed algorithms for DCOPs. Our three contributions are: (i) the development of a three–stage decomposition common to many completely distributed DCOP algorithms, (ii) the construction of five novel hybrids algorithms based on the algorithm decomposition, and (iii) an evaluation of these algorithms alongside six existing algorithms taken from the game theory and computer science literatures. Furthermore, our experimental results show that an algorithm's behaviour is accurately predicted by identifying its constituent components. Thus, using these results, a system designer may tailor a DCOP algorithm to suit their mix of requirements, whether they be high quality solutions, rapid convergence, asynchronous execution or low communication costs.

Future work involves examining dynamic constraint optimisation problems, in which constraints vary over time, both in response to external factors and as a result of the agents' choice of states. We also aim to extend the analysis in this paper to other types of problems in multi–agent systems, such as resource and task allocation problems and distributed management of congested networks.

6. ACKNOWLEDGEMENTS

This research was undertaken as part of the ALADDIN (Autonomous Learning Agents for Decentralised Data and Information Systems) project and is jointly funded by a BAE Systems and EPSRC strategic partnership. The authors would like to thank David Leslie for the many interesting discussions related to this work.

7. REFERENCES

[1] S. P. Anderson, A. de Palma, and J.-F. Thisse. *Discrete Choice Theory of Product Differentiation*. MIT Press, Cambridge, MA, 1992.

[2] G. Arslan, J. R. Marden, and J. S. Shamma. Autonomous vehicle-target assignment: A game theoretical formulation. *ASME Journal of Dynamic Systems, Measurement and Control*, 129:584–596, 2007.

[3] K. P. Bennett and E. Parrado-Hernández. The interplay of optimization and machine learning research. *Journal of Machine Learning Research*, 7:1265–1281, 2006.

[4] U. Berger. Brown's original fictitious play. *Journal of Economic Theory*, 135(1):572–578, 2007.

[5] G. W. Brown. Iterative solution of games by fictitious play. In T. C. Koopmans, editor, *Activity Analysis of Production and Allocation*, pages 374–376. Wiley, New York, 1951.

[6] S. Fitzpatrick and L. Meertens. Distributed coordination through anarchic optimization. In V. Lesser, C. L. O. Jr., and M. Tambe, editors, *Distributed Sensor Networks: A Multiagent Perspective*, pages 257–295. Kluwer Academic Publishers, 2003.

[7] D. Fudenberg and D. Kreps. Learning mixed equilibria. *Games and Economic Behavior*, 5:320–367, 1993.

[8] D. Fudenberg and D. K. Levine. *The Theory of Learning in Games*. MIT Press, Cambridge, MA, 1998.

[9] S. Hart and A. Mas-Colell. A simple adaptive procedure leading to correlated equilibrium. *Econometrica*, 68:1127–1150, 2000.

[10] M. Hayajneh and C. T. Abdallah. Distributed joint rate and power control game-theoretic algorithms for wireless data. *IEEE Communications Letters*, 8:511–513, 2004.

[11] T. Heikkinen. A potential game approach to distributed power control and scheduling. *Computer Networks*, 50:2295–2311, 2006.

[12] J. Hofbauer and W. H. Sandholm. On the global convergence of stochastic fictitious play. *Econometrica*, 70:2265–2294, 2002.

[13] R. T. Maheswaran, J. P. Pearce, and M. Tambe. A family of graphical-game-based algorithms for distributed constraint optimization problems. In *Coordination of Large-Scale Multiagent Systems*. Springer-Verlag, Heidelberg, Germany, 2005.

[14] D. Monderer and L. S. Shapley. Fictitious play property for games with identical interests. *Journal of Economic Theory*, 68:258–265, 1996.

[15] D. Monderer and L. S. Shapley. Potential games. *Games and Economic Behavior*, 14:124–143, 1996.

[16] A. Petcu and B. Faltings. DPOP: A scalable method for multiagent constraint optimization. In *IJCAI '05*, pages 266–271, Edinburgh, Scotland, Aug 2005.

[17] J. Robinson. An iterative method of solving a game. *Annals of Mathematics*, 54:296–301, 1951.

[18] R. W. Rosenthal. A class of games possessing pure-strategy Nash equilibria. *International Journal of Game Theory*, 2:65–67, 1973.

[19] T. Schiex, H. Fargier, and G. Verfaillie. Valued constraint satisfaction problems: Hard and easy problems. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI '95)*, pages 631–639, 1995.

[20] S. Singh, T. Jaakkola, M. L. Littman, and C. Szepesv́ari. Convergence results for single-step on-policy reinforcement-learning algorithms. *Machine Learning*, 39:287–308, 2000.

[21] M. Yokoo and K. Hirayama. Distributed breakout algorithm for solving algorithm for solving distributed constraint satisfaction and optimization problems. In *Proceedings of the Second International Conference on Multiagent Systems (ICMAS '96)*, pages 401–408, 1996.

[22] H. P. Young. *Individual Strategy and Social Structure: An Evolutionary Theory of Institutions*. Princeton University Press, New Jersey, 1998.

[23] W. Zhang and Z. Xing. Distributed breakout vs. distributed stochastic: A comparative evaluation on scan scheduling. In *Proceedings of the AAMAS '02 workshop on Distributed Constraint Reasoning*, 2002.

APPENDIX

A. ALGORITHM PSEUDOCODE

The following psuedocode decribes the algorithms used in our benchmarking experiments. First we give code for the six existing algorithms, followed by the five novel algorithms.

In all that follows, we denote an agent's neighbours' joint strategy profile s_{-i} and target function's value for strategy j as $stateValue(j)$ or $stateRegret(j)$, as appropriate. The pseudocode states the computations carried out by an individual agent, and unless otherwise stated, the algorithms are implemented by each agent running the stated procedure at each time step.

A.1 Existing Algorithms

The six existing algorithms listed discussed in Section 3.2 are the distributed stochastic algorithm, the maximum-gain messaging algorithm, fictitious play, smooth fictitious play, regret matching, and spatial adaptive play.

DISTRIBUTED STOCHASTIC ALGORITHM

currentValue = $u_i(s_i$= currentState, $s_{-i})$
for j = 1:J
  stateRegret(j) = $u_i(s_i{=}j, s_{-i})-$ currentValue
end for
candidateState = $\underset{j}{\text{argmax}}$ [stateRegret]
if rand[0,1] $\leq p$
  newState = candidateState
end if
if newState $\neq$ currentState
  sendStateMessage[allNeighbours, newState]
end if

MAXIMUM–GAIN MESSAGING

currentReward = $u_i(s_i$= currentState, $s_{-i})$
for j = 1:J
  stateGain(j) = $u_i(s_i{=}j, s_{-i})-$ currentReward
end for
bestGainState = $\underset{j}{\text{argmax}}$ [stateGain]
bestGainValue = stateGain(bestStateGain)
sendBestGainMessage[allNeighbours, bestGainValue]
neighbourGainValues = getNeighbourGainValues[allNeighbours]
if bestGainValue > max[neighbourGain] **then**
  newState = bestGainState
  sendStateMessage[allNeighbours, newState]
end if

FICTITIOUS PLAY

for j = 1:J
  stateValue(j) = $\frac{1}{t}[u_i(s_i{=}j, s_{-i}) + (t-1)$stateValue(j)$]$
end for
$t = t+1$
newState = $\underset{j}{\text{argmax}}$ [stateValue]
if newState $\neq$ currentState
  sendStateMessage[allNeighbours, newState]
end if

SMOOTH FICTITIOUS PLAY

for j = 1:J
  stateValue(j) = $\frac{1}{t}[u_i(s_i{=}j, s_{-i}) + (t-1)$stateValue(j)$]$
end for
$t = t+1$
for j = 1:J
  statePropensity(j) = exp[η^{-1}stateValue(j)]
end for
normFactor = $\sum_{j=1}^{J}$ statePropensity(j)
randomNumber = rand$(0,1)$
for j = 1:J
  mixedStrategyCDF(j) = $\frac{1}{\text{normFactor}}\sum_{k=1}^{j}$ statePropensity(k)
  if randomNumber $\leq$ mixedStrategyCDF(j) **then**
    newState = j
  break for **loop**
  end if
end for
if newState $\neq$ currentState
  sendStateMessage[allNeighbours, newState]
end if

REGRET MATCHING

currentValue = $u_i(s_i$= currentState, $s_{-i})$
for j = 1:J
  avgDiff(j) = $\frac{1}{t}(u_i(s_i{=}j, s_{-i})-$ currentValue $+(t-1)$avgDiff(j)$)$
  stateRegret(j) = max[avgDiff(j), 0]
end for
$t = t+1$
μ = (J$-$1)*(max$[u_i(j,l)] - \min[u_i(k,m)]) +$ rand$(0,1)$
for j = 1:J
  stateProbability(j) = $\frac{1}{\mu}$stateRegret(j)
end for
stateProbability(currentState) = 0
stateProbability(currentState) = $1-\sum_{j=1}^{J}$ stateProbability(j)
randomNumber = rand$(0,1)$
for j = 1:J
  mixedStrategyCDF(j) = $\sum_{k=1}^{j}$ stateProbability(k)
  if randomNumber $\leq$ mixedStrategyCDF(j) **then**
    newState = j
  break for **loop**
  end if
end for
if newState $\neq$ currentState
  sendStateMessage[allNeighbours, newState]
end if

In SAP, the agents adjust their state in a random sequence. In practice, there are a number of ways to implement this type of schedule, however, we simply randomly select an agent to run the stated procedure. For benchmarking purposes, we consider one time step to be completed when n (80) updates are completed. Note this can, and usually does, mean some agents may be given more than one opportunity to adjust their state in a particular time step, while other agents may have none.

SPATIAL ADAPTIVE PLAY

currentValue = $u_i(s_i$= currentState$, s_{-i})$
for j = 1:J
  stateRegret(j) = $u_i(s_i$=$j, s_{-i})-$ currentValue
end for
for j = 1:J
  statePropensity(j) = exp[η^{-1}stateRegret(j)]
end for
normFactor = $\sum_{j=1}^{J}$ statePropensity(j)
randomNumber = rand$(0,1)$
for j = 1:J
  mixedStrategyCDF(j) = $\frac{1}{\text{normFactor}}\sum_{k=1}^{j}$ statePropensity(k)
  if randomNumber $\leq$ mixedStrategyCDF(j) **then**
    newState = j
  break for **loop**
  end if
end for
if newState $\neq$ currentState
  sendStateMessage[allNeighbours, newState]
end if

A.2 Novel Hybrid Algorithms

The five novel hybrid algorithms described in Section 3.3 are: greedy spatial adaptive play; spatial fictitious play; distributed stochastic fictitious play; smooth spatial fictitious play; and maximum–gain message regret matching algorithm.

Regarding SFP and smSFP, these algorithms use the same agent procedure as the standard FP and smFP algorithms, respectively. Then, as in SAP, in SFP and smSFP the agents adjust their state in a random sequence, which is implemented as discussed above.

gSAP uses the same procedure as SAP to randomly order agents' state adjustments.

GREEDY SPATIAL ADAPTIVE PLAY

currentValue = $u_i(s_i$= currentState$, s_{-i})$
for j = 1:J
  stateRegret(j) = $u_i(s_i$=$j, s_{-i})-$ currentValue
end for
newState = $\underset{j}{\text{argmax}}$ [stateRegret]
if newState $\neq$ currentState
  sendStateMessage[allNeighbours, newState]
end if

The remaining algorithms are implemented by each agent running the stated procedure at each time step.

DISTRIBUTED STOCHASTIC FICTITIOUS PLAY

for j = 1:J
  stateValue(j) = $\frac{1}{t}[u_i(s_i$=$j, s^t_{-i}) + (t-1)$stateValue(j)]
end for
$t = t+1$
candidateState = $\underset{j}{\text{argmax}}$ [stateValue]
if rand[0,1] $\leq p$
  newState = candidateState
end if
if newState $\neq$ currentState
  sendStateMessage[allNeighbours, newState]
end if

MAXIMUM–GAIN MESSAGE REGRET MATCHING

currentValue = $u_i(s_i$= currentState$, s_{-i})$
for j = 1:J
  avgDiff(j) = $\frac{1}{t}(u_i(s_i$=$j, s_{-i})-$ currentValue $+(t-1)$avgDiff(j))
  stateRegret(j) = max[avgDiff(j), 0]
end for
$t = t+1$
μ = (J$-$1)*(max$[u_i(j,l)] -$ min$[u_i(k,m)])+$ rand$(0,1)$
for j = 1:J
  stateProbability(j) = $\frac{1}{\mu}$stateRegret(j)
end for
stateProbability(currentState) = 0
stateProbability(currentState) = $1-\sum_{j=1}^{J}$ stateProbability(j)
randomNumber = rand$(0,1)$
for j = 1:J
  mixedStrategyCDF(j) = $\sum_{k=1}^{j}$ stateProbability(k)
  if randomNumber $\leq$ mixedStrategyCDF(j) **then**
    candidateState = j
  break for **loop**
  end if
end for
candidateGainValue = $u_i(s_i$= candidateState$, s_{-i})-$ currentValue
sendGainMessage[allNeighbours, candidateGainValue]
neighbourGainValues = getNeighbourGainValues[allNeighbours]
if candidateGainValue $>$ max[neighbourGain] **then**
  newState = candidateState
  sendStateMessage[allNeighbours, newState]
end if