Auton Agent Multi-Agent Syst (2011) 22:385–414
DOI 10.1007/s10458-010-9128-3

Benchmarking hybrid algorithms for distributed constraint optimisation games

Archie C. Chapman · Alex Rogers · Nicholas R. Jennings

Published online: 16 April 2010

Abstract In this paper, we consider algorithms for *distributed constraint optimisation problems* (DCOPs). Using a potential game characterisation of DCOPs, we decompose eight DCOP algorithms, taken from the game theory and computer science literatures, into their salient components. We then use these components to construct three novel hybrid algorithms. Finally, we empirical evaluate all eleven algorithms, in terms of solution quality, timeliness and communication resources used, in a series of graph colouring experiments. Our experimental results show the existence of several performance trade-offs (such as quick convergence to a solution, but with a cost of high communication needs), which may be exploited by a system designer to tailor a DCOP algorithm to suit their mix of requirements.

Keywords 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. Within this context, in this paper, we focus specifically on *distributed constraint optimisation problems* (DCOPs), a broad family of problems that can be brought to bear on many domains, including: disaster response scenarios (e.g. [6,19]),

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.

A. C. Chapman (✉) · A. Rogers · N. R. Jennings
Electronics and Computer Science, University of Southampton, Southampton SO17 1BJ, UK
e-mail: acc@ecs.soton.ac.uk

A. Rogers
e-mail: acr@ecs.soton.ac.uk

N. R. Jennings
e-mail: nrj@ecs.soton.ac.uk

wide-area surveillance and distributed sensor network management (e.g. [14,15,18]), industrial task allocation and scheduling problems (e.g. [40,48]), and the management of congested air, road, rail, and information networks (e.g. [36,42]).

As a consequence of the breadth of applications of DCOPs, many algorithms for solving them have been developed using a number of approaches, which often differ according to the literatures they were first proposed in (e.g. the computer science, game theory, machine learning or statistical physics literatures). It is our intention, then, to investigate the properties of a broad class of DCOP algorithms. However, here we exclude from our analysis centralised approaches in which all of the information needed to solve the DCOP is directly accessible to, and/or in which all of the variables in a system come under the control of, a single decision maker, as assumed within algorithms such as the breakout algorithm [31] and arc consistency [7]. While such approaches are certainly useful in a range of scenarios, we make this exclusion because we are particularly interested in algorithms for multi-agent systems, in which the actors are distributed and can only communicate with their peers. The remaining algorithms are known as distributed algorithms, and, for our purposes, we define three further sub-groupings:

(i) *Distributed complete* algorithms, by which we mean algorithms that always find a configuration of variables that maximises the global objective function, such as ADOPT (Asynchronous Distributed OPTimization, [28]) or DPOP (Dynamic Programming OPtimisation, [32]), (ii) *Local iterative message-passing* algorithms, such as max-sum [1], in which neighbouring agents exchange messages comprising a data structure that contains the values of different local variable configurations, and use these values to construct new messages to pass on to other agents, and (iii) *Local iterative approximate best response* algorithms, such as the distributed stochastic algorithm [10,41] or fictitious play [5,33], in which agents exchange messages containing only their strategy, or can only observe the strategies of their neighbours. We refer to the last two groups together as local iterative algorithms to differentiate them from their complete counterparts. We group them under this term because both classes operate only at the local level, with messages exchanged between neighbouring agents at each iteration of the algorithm, and without any overarching structure controlling the timing or ordering of messages.

In many real distributed systems, we find that such local iterative algorithms are often preferred over distributed complete algorithms.[1] This is because, in such domains, *it is necessary and appropriate* to trade solution quality off against timeliness or communication overhead. For example, in real-time target tracking it may be more important to produce a good solution quickly, rather than wait for the optimal solution. This is the reasoning [20] invoke to motivate their use of the local iterative algorithm to coordinate scan schedules in a real meteorological radar network. Similarly, in remote and mobile sensor management, an algorithm that has a low communication overhead may be preferred because of the large drain on a sensor's battery charge caused by communication, as evidenced by the choice of algorithm used in many problems in distributed sensor networks (e.g. [8,47]) and multi-robot cooperative data fusion problems (e.g. [26,39]). Furthermore, in some real distributed systems, hardware limitations may outright prohibit the use of distributed complete algorithms. For example, when using DPOP the capacity of the communication buffer of a typical sensor node is quickly exceeded as a problem grows in size, because the message size is exponential in the induced width of the communication tree (e.g. [32,34]). Such exponential relationships are simply unacceptable in embedded devices that exhibit constrained computation,

[1] The complete algorithms are typically used in applications where their optimality is the key concern and timeliness is not a limiting factor, such as industrial scheduling and timetabling problems [32] or routing protocols for fixed environmental sensor networks [18].

bandwidth and memory resources. On the other hand, in these settings, it is clear that local algorithms are more effective, because the quality of the solutions they produce are typically satisfactory (even if they are not optimal), and they perform favourably in terms of the issues of scalability mentioned above.

Now, the characteristics of complete DCOP algorithms are well understood, and the properties of the entire framework of local message-passing algorithms have been extensively analysed, with guarantees placed on their solutions under a range of assumptions, for example, for tree structured [1] or single-looped constraint graphs [43]. In contrast, no such analysis has been completed for local iterative approximate best response algorithms. One reason for this is because, broadly speaking, these algorithms originate from two different literatures; either they are learning or adaptive processes taken from game theory, or they are distributed versions of centralised procedures developed for traditional constraint optimisation problems or heuristic search methods taken from computer science. In more detail, (centralised) constraint optimisation problems evolved, in part, as a method for analysing over-constrained constraint satisfaction problems. As such, traditional computer science approaches to such problems include the breakout algorithm, arc consistency, dynamic programming and stochastic optimisation techniques. Consequently, a traditional computer science approach to solving DCOPs, which includes many local approximate best response algorithms, starts by developing distributed versions of these centralised algorithms. For example, distributed breakout [16] and maximum-gain messaging [23] are two local approximate best response algorithms that descend from the breakout algorithm, and distributed versions of simulated annealing have been developed for DCOPs [3,10], which also fall into the category of local approximate best response algorithms. On the other hand, from a game-theoretic perspective, in a DCOP, each autonomous agent's aim is to maximise its own private utility function through its independent choice of strategy. From this point of view, each agent's optimal choice is strategically dependent on the actions of its neighbouring agents [22], and distributed algorithms for solving such problems is the focus of the literature of learning in games (e.g. [12]). However, what is common to both of these literatures is that the techniques used are all local, in that messages are only passed between neighbouring agents, iterative, in that a solution is generated by the repeated application of a particular decision-making process, approximate best response algorithms, in which agents exchange messages containing only their strategy. Typically, these algorithms converge to local optima (or Nash equilibria), and, moreover, it is game theory that has the tools and terminology to analyse algorithms that operate in such a manner. In particular, we stress that, in giving up global optimality, we consider the set of local optima, or equivalently, Nash equilibria, to be the appropriate solution concept for this class of algorithm. This is because this set represents the stable configurations of variables that can be reached by exchanging messages that contain only an agent's strategy (i.e. the information in the messages circulated in all of the algorithms in this class defines the appropriate solution concept).

Against this background, we devise an analytical framework, based on the class of potential games, to decompose eight of the major local iterative approximate best response algorithms for DCOPs into their constituent components: the distributed stochastic algorithm [10], the better-reply dynamics with inertia [27], spatial adaptive play [45], the maximum-gain messaging algorithm [23,44], fictitious play [5,33], smooth fictitious play [11,12], joint strategy fictitious play with inertia [25] and weighted regret monitoring with inertia [4]. We then use the components of the algorithms above to construct three novel DCOP algorithm hybrids. Our insight is that, by exploiting the convergence properties of potential games, we can construct these algorithms in a principled manner. Finally, we 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 seek to identify the effects that using various components have on the behaviour of an algorithm, such as identifying trade-offs for using different components (e.g. between the speed of convergence to a solution and its quality) or synergies that arise by using certain combinations of components.

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

2 DCOPs as potential games

In this section we state the key insight used in this paper: that an appropriately formulated DCOP game is a potential game. This result is used in subsequent sections to decompose existing algorithms into their constituent components and to guide the design of new algorithms. 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 analyse the algorithms' 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 (pure) *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. 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:

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

We can also define a *strict* Nash equilibrium by replacing the inequality with a strict inequality. Strict Nash equilibria play an important role in the convergence proofs of many of the algorithms considered in this paper.

The class of *potential games* is characterised as those games that admit a function specifying the participant's joint preference over outcomes [29]. This function is known as a potential function and, generally, it is a real-valued function on the joint strategy space (the Cartesian product of all agents' strategy spaces), defined such that the change in a unilaterally deviating player's utility is matched by the change in the potential function. The class of finite potential games have been used to describe many problems in multi-agent systems, in particular congestion problems on networks [35], and more recently, power control, channel selection and scheduling problems in wireless networks [14,15], target assignment problems [4] and job scheduling [48].

Formally, a function $P : S \to \mathbb{R}$ is an *(exact) potential* for a game if:

$$P(s_i, s_{-i}) - P\left(s_i', s_{-i}\right) = u_i(s_i, s_{-i}) - u_i\left(s_i', s_{-i}\right) \quad \forall\, s_i,\ s_i' \in S_i \quad \forall\, i \in N. \tag{2}$$

A game is called an *(exact) potential game* if it admits an exact potential. Informally, 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 [35]. 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 $i \in N$,

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

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 probability distribution over outcomes.

Second, every potential has the *finite improvement property* [30]. 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 Eq. 2:

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

Then, as S is a finite set, the sequence ϕ must be finite, so every potential game has the finite improvement property. This 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 framework for analysing DCOP algorithms in a principled manner.

A constraint optimisation problem is formally represented by a set of variables $X = \{x_1, \ldots x_n\}$, each of which may take one of a finite number of states or values, $s_i \in S_i$, 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 $c = \langle X_c, R_c \rangle$ is defined over a set of variables $X_c \subset X$ 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, $\prod_{x_j \in X_c} S_j$. 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 X_{c_k}. Using this, the global utility function aggregating the utilities from satisfying or violating constraints commonly takes the form:

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

Importantly, 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 [37]. The objective is then to find a global configuration of states, s^*, such that:

$$s^* \in \text{argmax}\, s \in S\; u_g(s).$$

Given this, a DCOP is produced when a set of autonomous agents each independently control the state of a subset of the variables, called the agent's *strategy*, but share the goal of maximising the rewards for satisfying constraints. A DCOP game is a simple formulation that explicitly models 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. We notate the set of agents involved in a constraint by N_c, the set of constraints containing i by C_i, and the agents that i shares constraints with, i's *neighbours*, by $\nu(i)$.

A DCOP game is formulated by assigning each agent a *private utility function*, $u_i(s)$, which is dependent on both its own strategy and the strategy 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 DCOP game in which the agents' private utilities are given by the sum of their constraint utilities, is a potential game.*

Proof Because a change in i's strategy only affects the utility of i's neighbours, $\nu(i)$, the following statements hold:

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

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

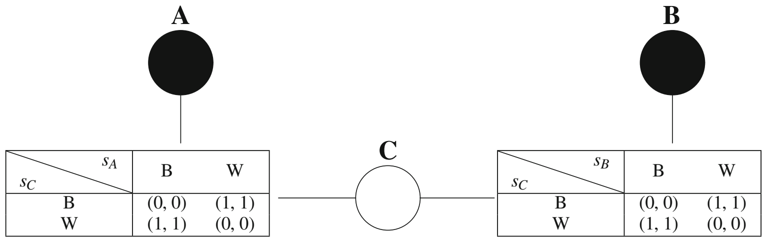
Although we focus on DCOPs in this paper, we now clarify the link between team games (the games used to model constraints) and the more general class of potential games. A team game (or common payoff game) is one in which every agent receives the same payoff for an outcome, and have a potential function equal to the common payoff function in the game. Any potential game can be expressed as a team game plus a *dummy game*, which is a game in which a player's payoff depends on its opponents' actions only (dummy games are also called the *non-strategic component* of a game). Importantly, the addition of a dummy game to a team game has no effect on the potential function of the game, because a change to an agent's action does not affect its payoff in the dummy game. Given this, the results (experimental and theoretical) that we present in this paper all relate to the potential function of the game, and consequently, the convergence proofs and experimental results hold for generalisations of team games and DCOPs to potential games.

In the case of binary constraints, the game played between pairs of agents sharing a constraint in a DCOP game can be easily expressed in bimatrix form, with the system of constraints represented in a simple graph. One widely studied binary constraint optimisation problem is graph colouring. Later in this paper we run a series of benchmarking experiments using graph colouring as our problem, and graph colouring is often viewed as a canonical example in the domain. For these reasons, we now 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 graph colouring problem in Fig. 1, where each node can be either black or white, and the associated 2 × 2 local game. Now, 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 local games it is playing with each neighbour. In Fig. 2, A and B are column players and C is the row player, payoffs (a) are (u_A, u_B, u_C), and a potential function for the game is below (b).

As the above example shows, complicated payoff structures may be constructed by combining simple constraint games.

Now, although the globally optimal joint pure strategy profile corresponds to a pure strategy Nash equilibrium, we emphasise that in most cases many Nash equilibria exist, and furthermore, many of those will be sub-optimal. However, with respect to the problem of designing control mechanism for large distributed systems, what is not specified in the above



s_C \ s_A	B	W
B	(0, 0)	(1, 1)
W	(1, 1)	(0, 0)

s_C \ s_B	B	W
B	(0, 0)	(1, 1)
W	(1, 1)	(0, 0)

Fig. 1 A three-node, two-colour graph colouring game

Fig. 2 **a** Payoff matrix and **b** potential for a graph colouring game

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)

A

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

B

formulation of a DCOP game are the processes by which agents adjust their strategies in order to arrive at an equilibrium. Indeed, there are many such processes, and in the next section we catalogue eight, and break them into their components, which are then used to construct novel DCOP algorithms.

3 Local iterative approximate best response algorithms for DCOPs

A comparison of local approximate best response algorithms for DCOPs is worthwhile in itself, however, here we are interested in 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 DCOP algorithms in this class. Then, in Sect. 3.2, we examine eight existing algorithms from the game theory and computer science literature, and fit them to our framework. These algorithms are the distributed stochastic algorithm (DSA), the better-reply dynamics with inertia (BR-I), spatial adaptive play (SAP), the maximum-gain messaging algorithm (MGM), fictitious play (FP), smooth fictitious play (smFP), joint strategy fictitious play with inertia (JSFP-I) and weighted regret monitoring with inertia (WRM-I). Finally, in Sect. 3.3 we construct three hybrid algorithms, using components taken from the eight existing algorithms. These hybrid algorithms are then evaluated alongside the eight existing algorithms in the experiments described in the Sect. 4.

3.1 The algorithm framework

The key feature identifying the class of local approximate best response algorithms is the information that is exchanged by the agents. In this class, agents exchange messages containing only their state, or can observe the strategies of their neighbours. In game theoretic parlance, this is known as *standard monitoring*, and, as the name suggests, is a typical informational assumption implicit in the literature on learning in games. Two contrasting classes of algorithms for DCOPs are *distributed complete* and *local iterative message-passing* algorithms. Distributed complete algorithms always find a configuration of variables that maximises the global objective function (as in finite domains one always exists) and includes ADOPT (Asynchronous Distributed OPTimization, [28]), DPOP (Dynamic Programming OPtimisation, [32]) and APO (Asynchronous Partial Overlay, [24]). In local iterative message-passing algorithms, such as max-sum [1] or distributed arc consistency [7], neighbouring agents exchange messages comprising a data structure that contains the values of different local variable configurations, and use these values to construct new messages to pass on to other agents. Algorithms in these two classes are not considered in this paper.

In more detail, in the DCOP algorithms we discuss, agents act myopically, in that they only consider the immediate effects of changing their strategies. Given an appropriate trigger, the individual agents follow the same basic two-stage routine, comprising *strategy evaluation*, which produces some measure of the desirability of each strategy, followed by a *decision* on which strategy to adopt, based on the preceding evaluations. The system-wide process that triggers an agent's internal processes is given by an *adjustment schedule*, that controls which agent adjusts its strategy at each point in time. In more detail:

Strategy evaluation: Each algorithm has a target function that it uses to evaluate its prospective strategies. The target functions are typically functions of payoffs, and sometimes take parameters that are set exogenously to the system or updated online. Additionally, some algorithms may make use of information about the past actions of agents to compute the value of the target function.
Decision rule: The decision rule refers to the procedure by which an agent uses its evaluation of strategies to decide upon an action to take. Typically an algorithm prescribes that an agent selects either the strategy that maximises (or minimises) the target function, or selects a strategy probabilistically, in proportion to the value of the target function for that strategy.
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. Given this, the communication requirements of each algorithm depend on the needs of these two stages. For example, algorithms that use random adjustment protocols only transfer information between agents to calculate the value of their target function (usually just their neighbours' strategies), whereas in algorithms that use preferential adjustment schedules (such as maximum-gain messaging), additional information may be required to run the adjustment schedule.

At this point, we make particular mention of two families of algorithms, MGM-k and SCA-k [23], that fall outside the class of algorithms we consider in this paper. In general, algorithms in these two families operate by agent's exchanging messages in order to compute the benefit of a joint or coordinated change in strategy by teams of up to size k. As such, the information exchanged by these algorithms exceeds the definition of the class of local approximate best response algorithms, for two reasons. First, the algorithms we consider in this paper communicate with their neighbours only. In contrast, the MGM-k and SCA-k algorithms assume an agent is able to communicate to any other agent on the constraint network. This is a clear violation of the definition of the algorithm class, and would make any conclusions drawn from direct comparisons of the experimental results somewhat spurious. Second, and more importantly, the amount of information exchanged using MGM-k or SCA-k is comparable or exceeds that used by message-passing algorithms like max-sum. To see this, note that in order to evaluate a joint change in action, agents share the gain in utility that they can earn for coordinating a change in their action with $k - 1$ team members. This is the same type of message passed my max-sum; a list of values, one for each potential action. As such, it is difficult to see why MGM-k would be used, because, like max-sum, it doesn't guarantee global optimisation, while at the same time it is more computationally intensive (due to the required comparisons of many possible teams' potential gains). In other words, although max-sum also doesn't guarantee global optimisation, typically it does converge to an optimal solution (see, e.g. [8]), and does so using less computation and similar communication requirements as MGM-k.

Before beginning the detailed discussion of specific algorithms, we define some of the terms we use in the analysis. In particular, we say an algorithm *almost surely converges* if the probability of the agents' joint strategy being a Nash equilibrium converges to 1 as the number of time steps tends to infinity. An algorithm *converges in distribution* if the distribution over joint strategies converges to some specified distribution as time tends to infinity. Typically, the specified distribution is the Boltzmann distribution over the joint strategy space with temperature η that maximises the global utility. Note that almost-sure convergence and convergence in distribution do not prevent the algorithm from moving arbitrarily far from a specified outcome, but only that these moves occur with decreasing probability [13]. An algorithm is called *anytime* if at each time step the solution it produces is at least as good as the one produced at the previous time step. Finally, a joint strategy is called *absorbing* if it is always played after the first time it is played, as is standard in the stochastic processes literature (see [13], for example).

We now discuss eight of the major iterative local approximate best response DCOP algorithms, and then construct three novel hybrid algorithms.

3.2 Existing algorithms

Using the framework described above, we now decompose eight main iterative local approximate best response DCOP algorithms into their constituent components, and examine how these components affect the algorithms' properties when they are used in repeated potential games. As an overview, Table 1 presents the algorithm components considered in this paper, Table 2 presents the parameterisation of the algorithms, and pseudocode for them is given in Appendix A.1.

3.2.1 Distributed stochastic algorithm

DSA uses the immediate payoff for selecting a strategy, $u_i(s_i, s_{-i})$ as its target function, and for a decision rule, uses the *argmax* function (i.e., selects the strategy 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 strategy at any time step [48]. This adjustment schedule is motivated by observing that when neighbouring agents adapt their strategies at the same time, they can inadvertently move their joint strategy 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, thrashing behaviour can be minimised.

Additionally, the finite improvement property ensures that DSA almost surely converges to a Nash equilibrium in repeated potential games (i.e. the probability that the system is in a Nash equilibrium converges to 1 as $t \to \infty$). Roughly, this is because at each time-step there is a positive probability of moving into a Nash equilibrium, which are absorbing when using the *argmax* decision rule and the immediate payoff as a target function.

Table 1 Components of the local approximate best response DCOP algorithms considered in this paper

Strategy evaluation	
Immediate payoff	$F_i^t(s_i) = u_i\left(s_i, s_{-i}^{t-1}\right)$
Fictitious play	$\mathrm{FP}_i^t(s_i) = \sum_{s_{-i} \in S_{-i}} \left(u_i(s_i, s_{-i}) \prod_{s_j \in s_{-i}} q_{s_j}^{t-1} \right)$
Joint strategy FP	$\mathrm{JSFP}_i^t(s_i) = \frac{1}{t}\left(u_i\left(s_i, s_{-i}^{t-1}\right) + (t-1) JSFP_i^{t-1}(s_i)\right)$
Weighted regret	$\mathrm{WR}_i^t(s_i) = \rho\left(u_i\left(s_i, s_{-i}^{t-1}\right) - u_i(s^{t-1})\right) + (1-\rho)\mathrm{WR}_i^{t-1}(s_i)$
Decision rule for any target function $F(s_i, s_{-i})$	
argmax-A	Current action selected if it satisfies $s_i^t = \mathrm{argmax}_{s_i \in S_i}$ [$F(s_i, s_{-i})$], otherwise a random element is selected from this set .
argmax-B	A random element from the set of actions satisfying $s_i^t = \mathrm{argmax}_{s_i \in S_i}$ [$F(s_i, s_{-i})$] is chosen .
Gain	Uniform choice from all actions $s_i' \in S_i$ for which $F\left(s_i', s_{-i}\right) - F\left(s_i^{t-1}, s_{-i}\right) \geq 0$.
Linear probability	An action is chosen with probability $P_{s_i^t} = \frac{F\left(s_i, s_{-i}^t\right)}{\sum_{s_i \in S_i} F\left(s_i, s_{-i}^t\right)}$
Logistic probability (η)	An action is chosen with probability $P_{s_i^t}(\eta) = \frac{e^{\eta^{-1} F\left(s_i, s_{-i}^t\right)}}{\sum_{s_i \in S_i} e^{\eta^{-1} F\left(s_i, s_{-i}^t\right)}}$
Adjustment schedule	
Flood	An agent moves at every time step.
Parallel random (p)	With probability $(1-p)$, an agent moves at a time step.
Maximum gain	If an agent can achieve the greatest gain out of all its neighbours, then it implements that change, otherwise it maintains its current strategy.
Sequential random	One agent changes per time step.

Table 2 Parameterisation of the main local approximate best response DCOP algorithms

	Target function	Decision rule	Adjustment schedule
DSA	Immediate payoff	argmax (A or B)	Parallel random (p)
MGM	Immediate payoff	argmax (A or B)	Maximum gain
BR-I	Immediate payoff	Gain	Parallel random (p)
SAP	Immediate payoff	Logistic (η)	Sequential random
FP	Fictitious play	argmax-B	Flood
smFP	Fictitious play	Logistic (η)	Flood
JSFP-I	Joint strategy FP	argmax-A	Parallel random (p)
WRM-I	Weighted regret	Linear prob$^+$ *or* logistic$^+$ (η)	Parallel random (p)

3.2.2 Better-reply dynamics with inertia

Like DSA, BR-I uses the immediate payoff for selecting a strategy as its target function and a random parallel adjustment schedule, but for a decision rule it selects any strategy that weakly improves its payoff, which we call a *gain* rule, rather than one that maximises its payoff. The proof of convergence of BR-I follows the same argument as that of DSA.

3.2.3 Maximum-gain messaging

Similar to DSA, MGM also uses immediate payoff and *argmax* function as its target function and decision rule, respectively. The algorithms differ purely by the adjustment schedule they employ: MGM uses a schedule that gives 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 strategy.

The maximum-gain messaging adjustment schedule avoids all thrashing, as no two neighbouring agents will ever move at the same time. As such, MGM always moves up the potential function to a local minimum, and therefore converges to a Nash equilibrium. Furthermore, since no thrashing occurs, it is an anytime algorithm, because the solution only ever weakly improves as the algorithm operates.

3.2.4 Spatial adaptive play

Like DSA, BR-I and MGM, SAP uses the immediate payoff for selecting a strategy as its target function. However, it differs from any of them in both its decision rule and adjustment schedule. SAP uses the multinomial logit decision rule [2], also known as the Boltzmann distribution:

$$P_{s_i}(\eta) = \frac{e^{\eta^{-1} u_i\left(s_i, s_{-i}^t\right)}}{\sum_{s_i \in S_i} e^{\eta^{-1} u_i\left(s_i, s_{-i}^t\right)}}. \tag{3}$$

Here strategies 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 strategy of the system following a random walk. The choice between the *argmax* decision rule and a probabilistic decision rule serves an important purpose. Using $\eta > 0$ 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 [38].

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 finite improvement property of potential games, which directly implies that agents who play a sequence of 'better responses' converge to a Nash equilibrium in a finite number of steps. This property, in conjunction with results regarding the convergence of GLIE processes, is used to prove the convergence of SAP to the global optimum of the potential function in repeated potential games [45, Chap. 6].[2] Note that this algorithm is closely related to *distributed simulated annealing* [3], but uses a different decision rule.

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

3.2.5 Fictitious play

In order to speed up convergence, an algorithm can use averaging to evaluate its strategies. One particular approach is to use the expected payoff for each strategy over historical frequencies of strategy profiles as a target function, and the term 'fictitious play' is often used to denote a family of adaptive processes which use this as a target function. These algorithms are constructed as follows. Let agent j's *historical frequency* of playing s_j' be defined as:

$$q_{s_j'}^t = \frac{1}{t}\sum_{\tau=0}^{t-1} I\left\{s_j' = s_j^\tau\right\},$$

where $I\{s_j' = s_j^\tau\}$ is an indicator function equal to one if s_j' is the strategy played by j at time τ, and zero otherwise. This may be stated recursively as:

$$q_{s_j'}^t = \frac{1}{t}\left[I\left\{s_j' = s_j^{t-1}\right\} + (t-1)q_{s_j'}^{t-1}\right].$$

Now, $q_{s_j'}^t$ may be interpreted as i's belief that its opponent, j, will play strategy s_j' at time t. Agent i's belief over each of its opponents' actions (i.e. for each $j \neq i$) is a vector of historical frequencies of play:

$$q_j^t = \begin{bmatrix} q_{s_j^1}^t \\ \vdots \\ q_{s_j^{|S_j|}}^t \end{bmatrix},$$

and i's belief over all of its opponents' actions is the set of vectors $q_{-i}^t = \{q_j^t\}_{j\in N\setminus i}$. Following this, i's expected payoff for playing s_i', given q_{-i}^t, is:

$$FP_i^t\left(s_i', q_{-i}^t\right) = \sum_{s_{-i}\in S_{-i}}\left[u_i\left(s_i', s_{-i}\right)\prod_{s_j\in s_{-i}} q_{s_j}^t\right] \tag{4}$$

where, in general, $S_{-i} = \cup_{j\neq i\in N} S_j$. However, note that in any DCOP game, q_{-i} and S_{-i} may be replaced with $q_{\nu(i)}$ and $S_{\nu(i)}$, respectively. In this paper, we consider the main version, often referred to as 'classical' FP, which uses the *argmax* decision rule (like DSA) in conjunction with the 'flood' adjustment schedule, under which all agents adjust their strategy simultaneously. Note that variations of fictitious play that use other methods to update the agent's belief state have been suggested, many of which are contained in the broad family of *generalised weakened* fictitious play processes [21].

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 agent's estimate of its opponents' strategies, which is used to calculate each of its own strategy's expected payoff, converges as time progresses [29]. Consequently, all agents' best response strategies 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.

3.2.6 Smooth fictitious play

smFP uses the expected payoff over historical frequencies of actions (Eq. 4) 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 (Eq. 3).

smFP converges to a Nash Equilibria, in much the same way as FP [17]. Nonetheless, in practice, using the logit decision rule does, on average, produce better quality solutions than the *argmax* rule. Leslie and Collins [21] show how to analyse smFP when the temperature parameter reduces over time and smFP approximates FPin the limit.

3.2.7 Joint strategy fictitious play with inertia

A similar infinite memory process to FP, JSFP was introduced by [25]. In this process, each agent keeps track of the frequency with which its opponents play each *joint* strategy s_{-i}, rather than their individual strategy frequencies. In this case, let i's belief over its opponents' joint strategy profiles, $q_i^t(s_{-i})$, be given by the fraction of times it observes each joint profile. Each agent's expected payoff given this belief is then:

$$\mathrm{JSFP}_i^t\left(s_i', q_{-i}^t\right) = \sum_{s_{-i}\in S_{-i}} q_i^t(s_{-i}) u_i\left(s_i', s_{-i}\right).$$

This can be expressed more simply as:

$$\mathrm{JSFP}_i^t\left(s_i', q_{-i}^t\right) = \frac{1}{t}\sum_{\tau=1}^{t} u_i\left(s_i', s_{-i}^\tau\right),$$

where s_{-i}^τ is the strategy profile of the agent's opponents at time τ. Furthermore, this target function can be specified recursively, which only requires agents to maintain a measure of the expected payoff for each strategy, rather than the full action history:

$$\mathrm{JSFP}_i^t\left(s_i', q_{-i}^t\right) = \frac{1}{t}\left[u_i\left(s_i', s_{-i}^t\right) + (t-1)\mathrm{JSFP}_i^{t-1}\left(s_i', q_{-i}^{t-1}\right)\right], \tag{5}$$

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.

Strict Nash equilibria are absorbing for any algorithm that uses the JSFP target function (Eq. 5) and *argmax* as a decision rule. This is because if agents have beliefs that induce them to play a strict Nash equilibrium, these beliefs are reinforced each time the strict Nash equilibrium is played. In particular, a version that operates on the parallel random schedule, JSFP *with inertia* (JSFP-I), converges to strict Nash equilibria [25], by the following argument. Given that strict Nash equilibria are absorbing, all that needs to be shown is that at any given time step, JSFP-I has some positive probability of visiting a strict Nash equilibrium. Now, under JSFP-I any unilateral change in strategy climbs the potential. Then, when inertia is added to the agents' choice of action (by using the parallel random adjustment schedule), the probability that a sequence of unilateral moves numbering at least the length of the longest improvement path occurs is strictly positive. Therefore, over time, the probability of entering the absorbing state approaches one.

3.2.8 Weighted regret monitoring with inertia

Another approach that can be used to speed up convergence is to measure the average 'regret' for not taking an action, where regret is the difference in payoff for choosing a strategy and the strategy that was actually chosen at a particular time. One particular regret-based algorithm is WRM-I, proposed by [4]. Under this process, past regrets are exponentially discounted, or the agents have 'fading memory':

$$\mathrm{WR}_i^t = \rho \left[u_i\left(s_i, s_{-i}^t\right) - u_i(s^t)\right] + (1-\rho)\mathrm{WR}_i^{t-1}, \tag{6}$$

where $(1-\rho)$ is the discount factor, $0 < \rho \leq 1$. WRM-I uses the parallel random schedule, and may be specified with any probabilistic decision rule that only selects a strategy from those with non-negative average regret (e.g. linear$^+$ or logit$^+$).

The proof of its convergence is similar to DSA and JSFP, and proceeds as follows. First, note that the target function used in WRM-I discounts past regrets (Eq. 6). As a consequence, if a given strict Nash equilibrium is played consecutively a sufficient number of times, it will be the only strategy for which any agent has a positive regret. Furthermore, the converse also holds: if each agent has only one strategy with non-negative regret, the corresponding joint strategy must be a Nash equilibrium. Second, the decision rules used in WRM-I only select from those strategies with non-negative regret. Therefore, if the joint regret-state is ever at a point where only one joint strategy has a positive regret for every agent, the algorithm will continue to select that joint strategy. Let us call this region in the agents' joint regret-space an equilibrium's *joint regret sink*. Third, the final step in the proof is to show that there is some strictly positive probability that the agents' joint regret enters an equilibrium's joint regret sink. This is achieved via the finite improvement property and the use of inertia, in an argument similar to that used to prove the convergence of DSA. Note that if past regrets are not discounted, then convergence to a Nash equilibrium cannot be guaranteed, and the algorithm may not even converge to a stationary point.

3.3 New hybrid algorithms

As noted in the introduction, we wish to find and exploit potential synergies that may arise from combinations of components from different existing DCOP algorithms. To this end, in this section, we investigate three hybrid algorithms constructed from the components identified previously. These particular hybrids were selected because, firstly, we expect them to display the types of synergies we are interested in, and secondly, because they validate our claim that the behaviour of an algorithm can be predicted by identifying which components are used in the algorithm.

Specifically, the first algorithm, *greedy spatial adaptive play*, removes the stochasticity in strategy choice from SAP. The second hybrid algorithm is *spatial joint strategy fictitious play*, which augments the joint strategy fictitious play target function with a new adjustment schedule. Finally, the third is *maximum-gain message weighted regret matching*, which uses maximum gain priority to schedule the agents' adjustment process. We note that many other combinations of algorithm components and parameter settings are possible, and indeed were examined. However, here we report only these three as they are the most interesting, the best performing, or give the clearest examples of how an algorithm's behaviour can, or cannot, be altered by changing a component. Table 3 presents the parameterisation of these algorithms, and pseudocode for them is given in Appendix A.2.

Table 3 Parameterisation of the main local approximate best response DCOP algorithms

	Target function	Decision rule	Adjustment schedule
gSAP	Immediate payoff	argmax	Sequential random
S-JSFP	Joint strategy FP	argmax	Sequential random
MGM-WRM	Weighted regret	Linear prob$^+$ *or* logitistic$^+$ (η)	Maximum gain

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 [4], and that by removing the stochasticity in strategy 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 joint strategy fictitious play

Spatial joint strategy fictitious play (S-JSFP) is a variation of JSFP in which the sequential adjustment schedule of SAP replaces the random parallel schedule of JSFP-I, while retaining the expected payoff over historical frequencies as the target function and $argmax$ as the decision rule.

Like gSAP and MGM, the convergence of this form of joint strategy fictitious play is a corollary of the finite improvement property of ordinal potential games. In more detail, under any form of JSFP(i.e. any algorithm that uses Eq. 5 as a target function), we note that if player i unilaterally switches from a_i to a_i', then there exists an improvement path from (a, a_{-i}) to (a_i', a_{-i}). In other words, the global utility is improved whenever an agent changes strategy in isolation. Now, under S-JSFP, only one agent changes strategy at a time. Therefore every sequence of moves traverses an improvement path, and consequently, the algorithm converges to a Nash equilibrium.

Besides its 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, so S-JSFP may be a better fit in many practical situations.

3.3.3 Maximum-gain message weighted regret monitoring

In the maximum-gain message weighted regret monitoring algorithm (MGM-WRM), agents employ the maximum-gain priority adjustment schedule in conjunction with the same target function and decision rule as WRM-I. In more detail, an agent selects a new strategy according to the combination of target function and decision rule, but only updates its strategy if it has the change in strategy that improves the global utility by the greatest amount among its neighbours.

Note that any algorithm using the maximum-gain priority adjustment schedule has the anytime property, because the algorithm will only move up the potential function towards a Nash equilibrium. Then, by using the combination of the weighted average regret target function (Eq. 6) and decision rules that only select from strategies with non-negative regret, the algorithm can never move down the potential function. Consequently, using this target function and decision rule only affects the path the algorithm takes up the potential function, and the subsequent equilibrium the algorithm comes to rest at, while the anytime property is maintained.

4 Empirical evaluation

We wish to test how the various 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. Furthermore, because graph colouring games are team games, they can be easily generalised to other forms of DCOPs and potential games by adding dummy games (as discussed in Sect. 2.2). The experimental results that we present in this section are all dependent on the potential function of the game in question, so as a consequence, these results are transferrable from graph colouring to other DCOPs and potential games more generally.

4.1 Experimental design

The set of graph colouring problems we consider is made up of two batches of 60 random graphs—one batch of 40 node graphs and one of 80 node graphs. Within each batch, 20 graphs are colourable in three colours, 20 in four colours and 20 in five colours. However, the number of colours (strategies) available to the agents is fixed at three, giving a mix of satisfiable and over-constrained problems to test our algorithms on. The mean connectivity (the average number of neighbours per node) of each graph is held constant at three for all the graphs considered. Preliminary experiments found that altering the mean connectivity of the graph had no effect on the relative quality of the solutions produced, with, as expected, the number of messages increasing linearly with the mean connectivity. Furthermore in the case that the number of neighbours is bounded (i.e. the problem is sparse), each agent faces a tractable computation. Specifically, since an agent has a bounded number of neighbours, the target function that it computes to evaluate its strategies takes a limited number of neighbours' actions as arguments. Consequently, the computational requirements of the algorithm is also bounded (typically to a polynomial factor with degree equal to the number of such inputs, with a specific form that varies across the target functions). If the connectivity of the problem has such a bound, then the local approximate best response algorithms considered in this paper will scale without difficulty, and the two batches of experiments we consider demonstrate this scalability.

In order to produce statistically significant results, 50 runs of each graph were completed by each algorithm. A run consisted of a maximum of 100 cycles, which was seen to be adequate for most of the algorithms to converge to a steady state. To put this in context, using MATLAB on a single machine with a 3.2 GHz Pentium$^{®}$ 4 processor and 1 GB of RAM, 50 runs of an algorithm on a 40 node graph took an average of 4.1 min to complete, and 5.5 min for the 80 node graphs, with the agents' processes run in series.

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. First, the ratio of the average utility of the solutions produced to the optimal, across all time steps, is used to broadly compare the overall performance of the algorithms. Second, the average of the ratio of the utility of the solution at the termination of the run to the optimal solution, is used to identify those algorithms that may be slow to converge but, in the end, provide high quality solutions. Third, 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 empirical 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 of the different algorithms, we plot the average utility against time.

4.2 Results and discussion

The simulation results are shown in Table 4, with 95% confidence intervals in parentheses. Where they are useful for illustrating some interesting behaviour of an algorithm, the relevant time series of the simulations are presented in Fig. 3. We begin our discussion by noting the overall similarity between the results for the 40 and 80 node graphs, with the exception of the proportion of terminal solutions that are optimal. This is evidence that the distributed computational approaches we consider are indeed scale free, as a technique that does not scale well would show a decline in the quality of its performance when faced with such an increase in the size of the problems considered. Consequently, we also expect that further increases in the size of the problems considered will have no detrimental effect on the quality of the solutions produced by the algorithms. The reason that the proportion of terminal solutions that are optimal generally decreases is simply that there are many more sub-optimal Nash equilibria in larger graphical games, thus a lower probability that an algorithm finds an optimal solution. We discuss this point in more detail in the context of the optimal algorithm SAP later.

We now continue with a detailed discussion of the results. We group the existing algorithms according to the type of target function they employ, beginning with the four algorithms that use immediate payoff or change in payoff: maximum-gain messaging algorithm, the distributed stochastic algorithm and, the better-reply dynamics with inertia and spatial adaptive play (MGM, DSA, BR-I and SAP, respectively). We then consider the four algorithms whose target functions employ averaging techniques: fictitious play, smooth fictitious play, joint strategy fictitious play with inertia and weighted regret monitoring with inertia (FP, smFP, JSFP-I and WRM-I, respectively). Then we relate results from the existing algorithms to those of the three novel hybrid algorithms discussed in Sect. 3.3: greedy spatial adaptive play, spatial joint strategy fictitious play and maximum-gain message weighted regret monitoring (gSAP, S-JSFP and MGM-WRM, respectively).[3]

[3] At this point, we note that we use a value of $p = 0.4$ for all of DSA, JSFP-I and WRM-I. We used $p = 0.4$ as a good representative of the spectrum of these algorithms as, for all three, it provided good quality solutions

Table 4 Graph colouring results for local approximate best response DCOP algorithms

Algorithm	Average utility	Ratio to optimum at T	Proportion optimal at T	Avg. time to converge	Avg. comm per node
40 node graphs					
MGM	0.912 (0.062)	0.922 (0.063)	0.03 (0.00)	53.5 (43.6)	1.55 (0.20)
DSA	0.963 (0.022)	0.984 (0.016)	0.29 (0.01)	12.7 (5.0)	0.09 (0.04)
BR-I	0.962 (0.022)	0.983 (0.017)	0.29 (0.01)	13.3 (5.1)	0.09 (0.04)
SAP	0.977 (0.016)	0.993 (0.009)	0.50 (0.01)	10.0 (3.4)	1.29 (0.12)
FP	0.943 (0.027)	0.952 (0.023)	0.02 (0.00)	8.5 (6.7)	0.07 (0.04)
smFP	0.930 (0.029)	0.959 (0.022)	0.04 (0.00)	32.6 (14.6)	0.38 (0.08)
JSFP-I	0.936 (0.025)	0.948 (0.024)	0.01 (0.00)	19.5 (14.2)	0.04 (0.03)
WRM-I	0.939 (0.025)	0.949 (0.023)	0.02 (0.00)	12.2 (4.2)	0.03 (0.03)
gSAP	0.979 (0.017)	0.987 (0.015)	0.36 (0.01)	3.0 (1.2)	0.89 (0.09)
S-JSFP	0.946 (0.024)	0.950 (0.024)	0.02 (0.00)	4.4 (3.7)	0.43 (0.08)
MGM-WRM	0.891 (0.044)	0.898 (0.043)	0.00 (0.00)	87.3 (31.2)	1.17 (0.23)
80 node graphs					
MGM	0.926 (0.039)	0.937 (0.039)	0.00 (0.00)	58.3 (41.3)	1.47 (0.27)
DSA	0.957 (0.017)	0.978 (0.015)	0.06 (0.00)	17.2 (6.1)	0.09 (0.06)
BR-I	0.957 (0.018)	0.978 (0.015)	0.06 (0.00)	17.8 (5.9)	0.09 (0.06)
SAP	0.974 (0.014)	0.991 (0.011)	0.25 (0.01)	16.1 (5.0)	1.32 (0.15)
FP	0.939 (0.019)	0.948 (0.018)	0.00 (0.00)	10.6 (7.8)	0.07 (0.05)
smFP	0.925 (0.021)	0.954 (0.018)	0.00 (0.00)	56.9 (20.6)	0.38 (0.11)
JSFP-I	0.933 (0.019)	0.945 (0.018)	0.00 (0.00)	25.4 (16.0)	0.03 (0.04)
WRM-I	0.935 (0.019)	0.946 (0.018)	0.00 (0.00)	14.5 (4.2)	0.03 (0.04)
gSAP	0.974 (0.015)	0.983 (0.014)	0.12 (0.01)	3.8 (1.4)	0.94 (0.11)
S-JSFP	0.941 (0.018)	0.946 (0.018)	0.00 (0.00)	6.3 (5.6)	0.45 (0.09)
MGM-WRM	0.897 (0.031)	0.903 (0.031)	0.00 (0.00)	97.2 (15.2)	1.11 (0.28)

4.2.1 Immediate reward and gain-based algorithm results

Regarding the existing algorithms, consider first those that use immediate rewards as their target function. In particular, in terms of solution quality, 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 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, and in part on the fact that the sequential adjustment schedule ensures no thrashing occurs. Note that the proportion of runs in which SAP finds an optimal solution drops off in the 80 node graph experiments. This is

Footnote 3 continued
in a timely manner at reasonable communication cost. Furthermore, the decision to use the same value for all three was made for the sake of brevity and to avoid clouding the discussion of algorithm *components* with a discussion of algorithm *parameters*. Nonetheless, we note that there is scope for investigating the possibility of setting p as a function of the number of neighbours an agent has in order to optimise the performance of an algorithm.

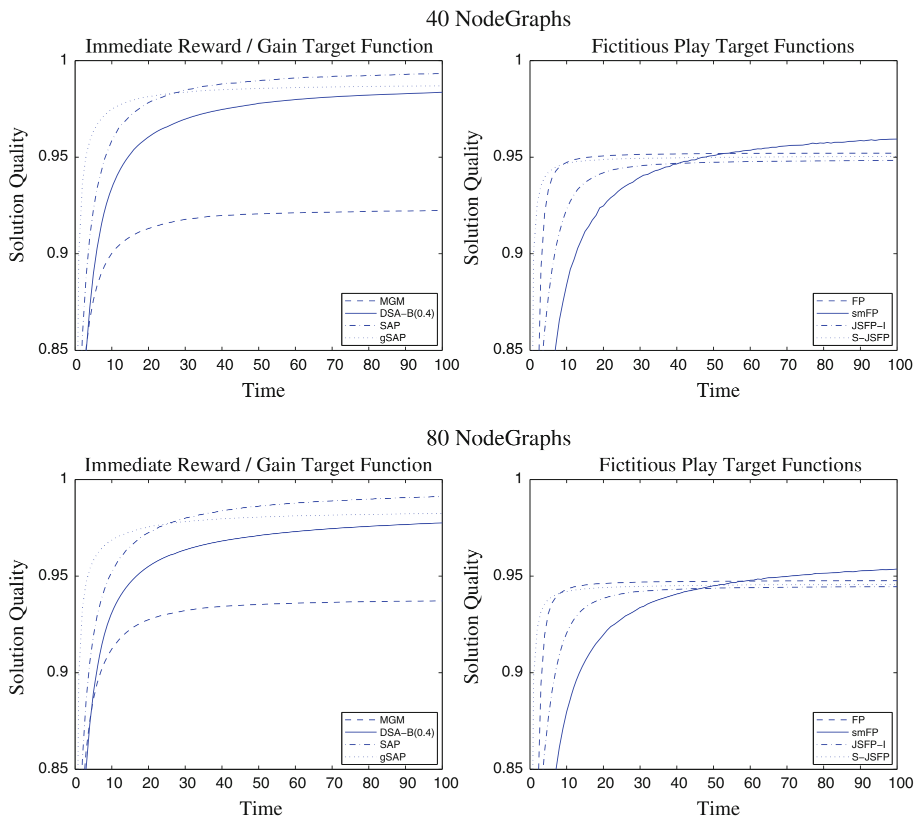


Fig. 3 Average utility versus time for local iterative approximate best response DCOP algorithms

because, even when the algorithms have found an optimal solution, the stochastic nature of the logistic decision rule acts to move the configuration of strategies away from the optimum with increasing probability as the graph becomes larger. However, we can conclude that this effect does not move the solution far from the optimum (that is, it does not induce many conflicts), because there is only a small variation in the ratio of the terminal solution's utility to the optimal utility between the 40 and 80 node results.

The two next-best performing algorithms—DSA and BR-I–share two common components: both use immediate rewards as a target function and the random parallel adjustment schedule, and both use decision rules that only select new strategies that are expected to improve the agent's utility. This explains the similarity in the quality of their solutions. Additionally, DSA and BR-I converge reasonably quickly because they allow parallel execution of strategy adjustments, and operate by communicating a relatively moderate number of messages.[4] While a direct comparison to the convergence time of SAP is not possible, we

[4] We make particular mention of the performance of DSA as a function of p in relation to results reported in [47]. Recall that p is set to 0.4 in our experiments. A marginal increase in p only very slightly improved the speed of convergence of DSA, with little or no improvement in the solution quality, and an increase in the communication requirements of the algorithm. Values of $p > 0.85$ show 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. These results are all consistent with those of [47].

see that SAP quickly finds a Nash equilibrium for the first time. However, SAP does use significantly more communication to achieve this benefit.

In contrast, MGM produces lower quality solutions than SAP, DSA or BR-I, and although it has similar communication requirements to SAP, it uses much more than DSA or BR-I, and takes much longer to converge. Furthermore, it rarely finds an optimal solution. These results are a consequence of using the maximum-gain adjustment schedule—MGM converges more slowly because it allows fewer agents to adjust at each time step,[5] and requires more messages to be communicated because the maximum gain messaging schedule is a two-stage adjustment process—and can be viewed as a trade-off for the anytime guarantee that is associated with this adjustment schedule.

4.2.2 Recursive averaging algorithm results

We now consider the performance of the algorithms using recursive averaging target functions. All three of FP, JSFP-I and WRM-I produce remarkably similar results in respect to the solution quality metrics, although these are all dramatically worse than the results for SAP, and less than DSA and BR-I. However, these losses may be traded-off against other metrics. For example, the convergence time of FP is the best of any of the existing algorithms considered in this paper. This also explains the slightly higher average utility generated by FP over JSFP-I and WRM-I. However, these benefits come at a cost of using twice as many messages per time step as JSFP-I and WRM-I,[6] which is of a level similar to DSA and BR-I. On the other hand, very little communication is used by JSFP-I and WRM-I, but their convergence time is still similar to DSA and BR-I. In particular, WRM-I produces good quality solutions using remarkably few messages. We suggest that, taken together, these results indicate that the recursive averaging target functions often select sub-optimal Nash equilibria, but can be used to improve either the convergence rate or communication requirement of an algorithm.

In contrast, smFP generates average utility results that are lower than FP, JSFP-I and WRM-I. This result is explained by smFP's use of the logit decision rule, which randomly perturbs the trajectory of the algorithm's convergence. Interestingly, the use of this decision rule improves the terminal solution quality and the proportion of terminal solutions that are optimal produced by smFP, which matches with the motivation for its use in conjunction with recursive averaging target functions. However, this benefit comes at a cost in terms of convergence time and communication requirement, both of which are significantly greater than the results for FP, JSFP-I and WRM-I, as well as DSA and BR-I.

4.2.3 The novel hybrid algorithm results

We now consider the novel hybrid algorithms, and relate their performance to that of the existing algorithms. Our aims in doing so are to identify synergies flowing from novel combinations of techniques, to test if the specific results above have more general applicability, and to see if the predictions we made in Sect. 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

[5] Note that the average times to converge given in Table 4 is likely to be an underestimate, because the metric is biased by truncating its value at 100 time steps.

[6] Note that this is not a reflection of the choice of $p = 0.4$ for JSFP-I and WRM-I, whose behaviour show similar patterns to that of DSA in response to variations in p.

terms of decreased terminal solution quality and proportion of terminal solutions that are optimal. This is expected. However, as shown in Fig. 3, the trade-off is quite one-sided, with the average solution produced by gSAP overtaking SAP's solution early in the simulation. Furthermore, longer simulations showed that the solutions produced by SAP continued to improve, while the average solution quality produced by gSAP remained at about the level seen in Table 4. 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, a result flowing from both its convergence to Nash equilibrium and its use of the *argmax* decision rule. Consequently, gSAP is considered useful if there is a very short time horizon or convergence to a Nash equilibrium is a necessity (recalling that SAP only converges almost-surely).

Second, S-JSFP produces solutions of similar quality to FP, JSFP-I and WRM-I, but converges in significantly fewer time steps. Specifically, the algorithm converges in between 60% of the number of time steps used by FP and 25% of those used by JSFP-I. Now, although using a random sequence of moves does nothing to alter the speed of convergence of the beliefs used in this algorithms' target function *per se*, in the early stages of the run, this adjustment schedule does act to reduce any thrashing that may occur before beliefs have begun to converge. This particular effect of using the random sequential adjustment schedule is also seen in the comparison of gSAP to BR-I or DSA. Overall, this is a positive result, because no loss of performance implies that FP, JSFP-I or S-JSFP 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, or the time available for the algorithm to converge to a solution.

Third, regarding MGM-WRM, using the maximum-gain priority schedule did little to improve the quality of the solutions produced by using weighted average regret as a target function and the linear probabilistic decision rule. This is because the strategies 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. This behaviour is observed in spite of the fact that the algorithm is guaranteed to converge to a Nash equilibrium, because the guarantee is that the algorithm will eventually move away from non-equilibria, and does not give any assurances as to when this will happen. In fact, given that the convergence metric is truncated at 100, the average times to converge given in Table 4 indicate that the algorithm rarely found a Nash equilibrium. Additionally, the maximum-gain priority schedule uses a significantly greater number of messages to run than the other algorithms considered, with the exception of MGM.

4.3 Summary

In summary, the main effects of using different algorithm components are as follows.

1. 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.
2. The sequential random adjustment schedule always improves the convergence time of an algorithm, but for target functions other than immediate payoffs or gain, it does not improve solution quality.

3. Due to the averaging mechanism they employ, algorithms that use the expected payoff over historical frequencies of actions as a target function—FP, JSFP-I and S-JSFP—converge quickly and use few messages, because few time steps are wasted in thrashing and cycling behaviours.
4. Using an ergodic stochastic decision rule (i.e. multinomial logit) in conjunction with either an immediate reward target function (as in SAP) or a fictitious play-type target function (as in smFP) does improve the final solution quality, but at the cost of increasing both the convergence time and communication costs of the algorithm. In the case of smFPthis is interesting because, even though it is intuitively appealing to think that smFP should find higher payoff solutions more frequently than other, non-ergodic algorithms, it is only guaranteed to converge to an arbitrary element in the set of 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, S-JSFP and, and on the other side, MGM, BR-I, JSFP-I, WRM-I and MGM-WRM). In 100 cycles, gSAP provides $100 \times n$ individual adjustment opportunities to agents in the system. DSA, on the other hand, only provides $p \times 100 \times n$ adjustment opportunities (in the case of $p = 0.4$, $40 \times n$). 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 $t = 40$ to the average solution quality of DSA(0.4) at $t = 80$, we find in the 40 node case an average ratio of utility to optimum of 0.968 for gSAP compared with 0.963 for DSA(0.4), and in the 80 node case values of 0.957 and 0.962, 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 $0.4 \times 12.7 = 5.08$. 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.

5 Conclusions

In this paper, we focused on iterative local approximate best response algorithms for DCOPs. The paper reported results from a series of graph colouring experiments, which were used to benchmark a suite of eight existing and three novel hybrid DCOP algorithms. Our experimental results showed that an algorithm's behaviour can be accurately predicted by identifying its constituent components. Thus, using our framework, a system designer may tailor a DCOP algorithm to suit their mix of requirements, whether they are high quality solutions, rapid convergence, asynchronous execution or low communication costs. We believe our results reflect the general properties of the class of problems addressed, and so expect them to generalise to all problems that may be represented by potential games in which the optimal joint strategy resides in the set of Nash equilibria. As evidence for this claim, we note that results consistent with ours have been reported in a task-assignment problem modelled as a potential game [4], and in other graph colouring and more general constraint satisfaction experiments [3,9,23,46].

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 (Engineering and Physical Sciences Research Council) strategic partnership (EP/C548051/1).

Appendix A: Local approximate best response algorithm pseudocode

The following pseudocode describes the algorithms used in our benchmarking experiments. First we give code for eight existing algorithms (from Sect. 3.2), followed by the three novel ones (as detailed in Sect. 3.3).

The pseudocode states the computations carried out by an individual agent, and unless otherwise stated, the algorithms (including their various adjustment schedules) are implemented by each agent running the stated procedure at every time step. In all that follows, we drop the subscript i because the pseudocode refers to an agent's internal processes. We denote an agent's strategy $s \in S$ and its target function's value for strategy k as *stateValue(k)* or *stateRegret(k)*, as appropriate. An agent's neighbours are indexed $j \in \nu$, with their joint strategy profile notated s_ν. Finally, an agent's immediate payoff for an strategy k, given its neighbours' joint strategy profile is written $u(k, s_\nu)$.

A.1: Existing algorithms

The eight existing algorithms listed here are discussed in Sect. 3.2, and are the maximum-gain messaging algorithm (MGM), the distributed stochastic algorithm using the $argmax$-B decision rule (DSA), better-response with inertia (BR-I), spatial adaptive play (SAP), fictitious play (FP), smooth fictitious play (smFP), joint strategy fictitious play with inertia (JSFP-I) and weighted regret monitoring with inertia (WRM-I).

Maximum-gain messaging (MGM)

```
currentReward = $u(s=$ currentState, $s_\nu)$                        1
for k = 1:K                                                          2
  stateGain(k) = $u(s=k, s_\nu)$– currentReward                      3
end for                                                              4
bestGainState = argmax $k$[stateGain]                                5
bestGainValue = stateGain(bestStateGain)                             6
sendBestGainMessage[allNeighbours, bestGainValue]                    7
neighbourGainValues = getNeighbourGainValues[allNeighbours]          8
if bestGainValue > max[neighbourGain] then                           9
  newState = bestGainState                                          10
  sendStateMessage[allNeighbours, newState]                         11
end if                                                              12
```

Distributed stochastic algorithm (DSA)

```
currentValue = $u(s=$ currentState, $s_\nu)$                         1
for k = 1:K                                                          2
  stateRegret(k) = $u(s=k, s_\nu)$– currentValue                     3
end for                                                              4
candidateState = argmax $k$[stateRegret]                             5
if rand[0,1] $\leq p$                                                6
  newState = candidateState                                          7
end if                                                               8
if newState $\neq$ currentState                                      9
  sendStateMessage[allNeighbours, newState]                         10
end if                                                              11
```

Better-response with inertia (BR-I)

```
currentValue = $u$($s$= currentState, $s_\nu$)
for k = 1:K
  stateRegret(k) = max[$u$($s$=$k$, $s_\nu$)– currentValue,0]
end for
normFactor = $\sum_{k=1}^{K}$ stateRegret
randomNumber = rand(0, 1)
for k = 1:K
  mixedStrategyCDF(k) = $\frac{1}{\text{normFactor}} \sum_{l=1}^{k}$ stateRegret(l)
  if randomNumber $\leq$ mixedStrategyCDF(k) then
    candidateState = k
  break for loop
  end if
end for
if rand[0,1] $\leq p$
  newState = candidateState
end if
if newState $\neq$ currentState
  sendStateMessage[allNeighbours, newState]
end if
```

In SAP the agents adjust their strategy 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 updates are completed (in our experiments, n is 40 or 80). Note this can, and usually does, mean some agents may be given more than one opportunity to adjust their strategy in a particular time step, while other agents may have none. See Sect. 4.3 for a discussion of the effect this has on the metrics used.

Spatial adaptive play (SAP)

```
currentValue = $u$($s$= currentState, $s_\nu$)
for k = 1:K
  stateRegret(k) = $u$($s$=$k$, $s_\nu$)– currentValue
end for
for k = 1:K
  statePropensity(k) = exp[$\eta^{-1}$stateRegret(k)]
end for
normFactor = $\sum_{k=1}^{K}$ statePropensity(k)
randomNumber = rand(0, 1)
for k = 1:K
  mixedStrategyCDF(k) = $\frac{1}{\text{normFactor}} \sum_{l=1}^{k}$ statePropensity(l)
  if randomNumber $\leq$ mixedStrategyCDF(k) then
    newState = k
  break for loop
  end if
end for
if newState $\neq$ currentState
  sendStateMessage[allNeighbours, newState]
end if
```

In FP and smFP, $|\nu|$ is the number of neighbours an agent has, $\mathbf{q}_j$ is a vector of the frequencies with which neighbour j has played each strategy k_j in the past, $I\{k_j = s_j^t\}$ is an indicator vector with an element equal to one for the strategy k_j played by j at time t and zero everywhere else, and $H = \prod_{j=1}^{|nu|} |S_j|$ is the size of the agent's neighbours' joint strategy space.

Fictitious play (FP)

```
for j = 1:$|\nu|$
  $\mathbf{q}_j^t = \frac{1}{t}[I\{k_j = s_j^t\} + (t-1)\mathbf{q}_j^{t-1}]$
end for
$t = t + 1$
for k = 1:K
  for h = 1:H
    $\mathbb{E}[u(s = k, s_\nu^h)] = u(s, s_\nu^h) \prod_{s_j^h \in s_\nu^h} q_j^t$
  end for
  stateValue(k) = $\sum_{h=1}^{H} \mathbb{E}[u(s = k, s_\nu^h)]$
end for
newState = argmax $k$[stateValue]
if newState $\neq$ currentState
  sendStateMessage[allNeighbours, newState]
end if
```

Smooth fictitious play (smFP)

```
for j = 1:$|\nu|$
  $\mathbf{q}_j^t = \frac{1}{t}[I\{k_j = s^t\} + (t-1)\mathbf{q}_j^{t-1}]$
end for
$t = t + 1$
for k = 1:K
  for h = 1:H
    $\mathbb{E}[u(s = k, s_\nu^h)] = u(s, s_\nu^h) \prod_{s_j^h \in s_\nu^h} q_j^t$
  end for
  stateValue(k) = $\sum_{h=1}^{H} \mathbb{E}[u(s = k, s_\nu^h)]$
end for
for k = 1:K
  statePropensity(k) = exp[$\eta^{-1}$stateValue(k)]
end for
normFactor = $\sum_{k=1}^{K}$ statePropensity(k)
randomNumber = rand(0, 1)
for k = 1:K
  mixedStrategyCDF(k) = $\frac{1}{\text{normFactor}} \sum_{l=1}^{k}$ statePropensity(l)
  if randomNumber $\leq$ mixedStrategyCDF(k) then
    newState = k
  break for loop
  end if
end for
if newState $\neq$ currentState
  sendStateMessage[allNeighbours, newState]
end if
```

Joint strategy fictitious play with inertia (JSFP-I)

```
for k = 1:K
  stateValue(k) = $\frac{1}{t}[u(s{=}k, s_\nu) + (t-1)$stateValue(k)]
end for
$t = t + 1$
candidateState = argmax $k$[stateValue]
if rand[0,1] $\leq p$
  newState = candidateState
end if
if newState $\neq$ currentState
  sendStateMessage[allNeighbours, newState]
end if
```

Weighted regret matching with inertia (WRM-I)

```
currentValue = $u(s$= currentState, $s_\nu)$                                  1
for k = 1:K                                                                   2
  avgDiff(k) = $\rho u(s{=}k, s_\nu$– currentValue + $(1 - \rho)$avgDiff(k))   3
  stateRegret(k) = max[avgDiff(k),0]                                          4
end for                                                                       5
normFactor = $\sum_{k=1}^{K}$ stateRegret                                     5
randomNumber = rand(0, 1)                                                     6
for k = 1:K                                                                   7
  mixedStrategyCDF(k) = $\frac{1}{\text{normFactor}} \sum_{l=1}^{k}$ stateRegret(l)   8
  if randomNumber $\leq$ mixedStrategyCDF(k) then                             12
    candidateState = k                                                        9
  break for loop                                                              10
  end if                                                                      11
end for                                                                       12
if rand[0,1] $\leq p$                                                         13
  newState = candidateState                                                   14
end if                                                                        15
if newState $\neq$ currentState                                               16
  sendStateMessage[allNeighbours, newState]                                   17
end if                                                                        18
```

A.2: Novel hybrid algorithms

The three novel hybrid algorithms described in Sect. 3.3 are: greedy spatial adaptive play (gSAP); spatial joint strategy fictitious play (S-JSFP); and maximum-gain message weighted regret matching (MGM-WRM).

gSAP uses the same procedure as SAP to randomly order agents' strategy adjustments. The novelty is in the use of the *argmax* function to assign a value to *newState*, as shown on line 5, rather than the multinomial logit (lines 5 to 16 in SAP).

Greedy spatial adaptive play (gSAP)

```
currentValue = $u(s$= currentState, $s_\nu)$                 1
for k = 1:K                                                  2
  stateRegret(k) = $u(s{=}k, s_\nu)$– currentValue           3
end for                                                      4
newState = argmax $k$[stateRegret]                           5
if newState $\neq$ currentState                              6
  sendStateMessage[allNeighbours, newState]                  7
end if                                                       8
```

Regarding S-JSFP, this algorithm uses the same target function as the JSFP-I. The novelty is that in S-JSFP, the agents adjust their strategy in a random *sequence*, which is implemented as discussed the context of SAP. The procedure run by each agent is given below.

Spatial joint strategy fictitious play (SJSFP)

```
for k = 1:K                                                              1
  stateValue(k) = $\frac{1}{t}[u(s{=}k, s_\nu) + (t - 1)$stateValue(k)]   2
end for                                                                  3
$t = t + 1$                                                              4
newState = argmax $k$[stateValue]                                        5
if newState $\neq$ currentState                                          6
  sendStateMessage[allNeighbours, newState]                              7
end if                                                                   8
```

In MGM-WRM, the maximum-gain priority adjustment schedule is used by agents who select their new strategies by the same method as those using WRM-I. In comparison to WRM-I, this hybridisation can be seen in the lines after line 13 of the pseudocode below.

Maximum-gain message regret matching (MGM-WRM)

```
currentValue = $u(s=$ currentState, $s_\nu)$                                      1
for k = 1:K                                                                   2
  avgDiff(k) = $\rho u(s=k, s_\nu -$ currentValue $+ (1-\rho)$avgDiff(k))          3
  stateRegret(k) = max[avgDiff(k), 0]                                         4
end for                                                                       5
normFactor = $\sum_{k=1}^{K}$ stateRegret                                        5
randomNumber = rand(0, 1)                                                     6
for k = 1:K                                                                   7
  mixedStrategyCDF(k) = $\frac{1}{\text{normFactor}} \sum_{k=1}^{k}$ stateRegret(k)   8
  if randomNumber $\leq$ mixedStrategyCDF(k) then                               12
    candidateState = k                                                        9
  break for loop                                                              10
  end if                                                                      11
end for                                                                       12
candidateGainValue =$u(s=$ candidateState, $s_\nu)-$ currentValue                13
sendGainMessage[allNeighbours, candidateGainValue]                            14
neighbourGainValues = getNeighbourGainValues[allNeighbours]                   15
if candidateGainValue > max[neighbourGain] then                               16
  newState = candidateState                                                   17
  sendStateMessage[allNeighbours, newState]                                   18
end if                                                                        19
```

References

1. Aji, S. M., & McEliece, R. J. (2000). The generalized distributive law. *IEEE Transactions on Information Theory, 46*, 325–343.
2. Anderson, S. P., de Palma, A., & Thisse, J. (1992). *Discrete choice theory of product differentiation*. Cambridge: MIT Press.
3. Arshad, M., & Silaghi, M. C. (2003). Distributed simulated annealing and comparison to DSA. In *Proceedings of the fourth international workshop on distributed constraint reasoning (DCR-03)*. Acapulco, Mexico.
4. Arslan, G., Marden, J. R., & Shamma, J. S. (2007). Autonomous vehicle-target assignment: A game theoretical formulation. *ASME Journal of Dynamic Systems, Measurement and Control, 129*, 584–596.
5. Brown, G. W. (1951). Iterative solution of games by fictitious play. In T. C. Koopmans (Ed.), *Activity analysis of production and allocation* (pp. 374–376). New York: Wiley.
6. Chapman, A., Micillo, R. A., Kota, R., & Jennings, N. (2009). Decentralised dynamic task allocation: A practical game-theoretic approach. In *The 8th international conference on autonomous agents and multiagent systems (AAMAS-09)*, pp. 915–922.
7. Cooper, M., de Givry, S., & Schiex, T. (2007). Optimal soft arc consistency. In *Proceedings of the 20th internation joint conference on artificial intelligence (IJCAI-07)*, pp. 68–73.
8. Farinelli, A., Rogers, A., Petcu, A., & Jennings, N. R. (2008). Decentralised coordination of low-power embedded devices using the max-sum algorithm. In *Proceedings of the 7th international conference on autonomous agents and multiagent systems (AAMAS-08)*, pp. 639–646.
9. Fitzpatrick, S., & Meertens, L. (2001). An experimental assesment of a stochastic, anytime, decentralized soft colourer for sparse graphs. In K. Steinhoefel (Ed.), *Stochastic algorithms: Foundations and applications, Proceedings SAGA-1* (pp. 49–64). Berlin: Springer.
10. Fitzpatrick, S., & Meertens, L. (2003). Distributed coordination through anarchic optimization. In V. Lesser, C. L. Ortiz, Jr., & M. Tambe (Eds.), *Distributed sensor networks: A multiagent perspective* (pp. 257–295). Boston: Kluwer.
11. Fudenberg, D., & Kreps, D. (1993). Learning mixed equilibria. *Games and Economic Behavior, 5*, 320–367.
12. Fudenberg, D., & Levine, D. K. (1998). *The Theory of learning in games*. Cambridge: MIT Press.

13. Grimmett, G., & Stirzaker, D. (2001). *Probability and random processes* (3rd ed.). Oxford: Oxford University Press.
14. Hayajneh, M., & Abdallah, C. T. (2004). Distributed joint rate and power control game-theoretic algorithms for wireless data. *IEEE Communications Letters, 8*, 511–513.
15. Heikkinen, T. (2006). A potential game approach to distributed power control and scheduling. *Computer Networks, 50*, 2295–2311.
16. Hirayama, K., & Yokoo, M. (2005). The distributed breakout algorithms. *Artificial Intelligence, 161*(1-2), 89–115.
17. Hofbauer, J., & Sandholm, W. H. (2002). On the global convergence of stochastic fictitious play. *Econometrica, 70*, 2265–2294.
18. Kho, J., Rogers, A., & Jennings, N. R. (2009). Decentralised control of adaptive sampling in wireless sensor networks. *ACM Transactions on Sensor Networks, 5*(3), 19–35.
19. Kitano, H., Todokoro, S., Noda, I., Matsubara, H., Takahashi, T., Shinjou, A., & Shimada, S. (1999). Robocup rescue: Search and rescue in large-scale disaster as a domain for autonomous agents research. In *Proceedings of the IEEE international conference on system, man, and cybernetics (SMC-99)*, Vol. 6, pp. 739–743. Tokyo, Japan.
20. Krainin, M., An, B., & Lesser, V. (2007). An application of automated negotiation to distributed task allocation. In *Proceedings of the 2007 IEEE/WIC/ACM international conference on intelligent agent technology (IAT-07)*, pp. 138–145. Fremont, CA: IEEE Computer Society Press.
21. Leslie, D. S., & Collins, E. J. (2006). Generalised weakened fictitious play. *Games and Economic Behavior, 56*, 285–298.
22. Maheswaran, R. T., Pearce, J. P., & Tambe, M. (2004) Distributed algorithms for DCOP: A graphical-game-based approach. In *Proceedings of the 17th international conference on parallel and distributed computing systems (PDCS-04)*, pp. 432–439, San Francisco, CA, USA.
23. Maheswaran, R. T., Pearce, J. P., & Tambe, M. (2005). A family of graphical-game-based algorithms for distributed constraint optimization problems. In *Coordination of large-scale multiagent systems*, pp. 127–146. Heidelberg: Springer.
24. Mailler, R., & Lesser, V. (2006). Asynchronous partial overlay: A new algorithm for solving distributed constraint satisfaction problems. *Journal of Artificial Intelligence Research, 25*, 529–576.
25. Marden, J. R., Arslan, G., & Shamma, J. S. (2009). Joint strategy fictitious play with inertia for potential games. *IEEE Transaction on Automatic Control, 54*(2), 208–220.
26. Matthews, G. M., & Durrant-Whyte, H. F. (2006). Scalable decentralised control for multi-platform reconnaissance and information gathering tasks. In *Proceedings of the 9th international conference on information fusion (Fusion'06).*
27. Mezzetti, C., & Friedman, J. W. (2001). Learning in games by random sampling. *Journal of Economic Theory, 98*(1), 55–84.
28. Modi, P. J., Shen, W., Tambe, M., & Yokoo, M. (2005). ADOPT: Asynchronous distributed constraint optimization with quality guarantees. *Artificial Intelligence, 161*(1-2), 149–180.
29. Monderer, D., & Shapley, L. S. (1996). Fictitious play property for games with identical interests. *Journal of Economic Theory, 68*, 258–265.
30. Monderer, D., & Shapley, L. S. (1996). Potential games. *Games and Economic Behavior, 14*, 124–143.
31. Morris, P. (1993). The breakout method for escaping from local minima. In *Proceedings of the 11th national conference on artificial intelligence (AAAI '93)*, pp. 40–45. Washington, DC, USA.
32. Petcu, A., & Faltings, B. (2005). DPOP: A scalable method for multiagent constraint optimization. In *Proceedings of the 19th international joint conference on artificial intelligence (IJCAI-05)*, pp. 266–271. Edinburgh, Scotland.
33. Robinson, J. (1951). An iterative method of solving a game. *Annals of Mathematics, 54*, 296–301.
34. Rogers, A., Corkill, D. D., & Jennings, N. R. (2009). Agent technologies for sensor networks. *IEEE Intelligent Systems, 24*(2), 13–17.
35. Rosenthal, R. W. (1973). A class of games possessing pure-strategy Nash equilibria. *International Journal of Game Theory, 2*, 65–67.
36. Roughgarden, T. (2005). *Selfish routing and the price of anarchy*. Cambridge: MIT Press.
37. Schiex, T., Fargier, H., & Verfaillie, G. (1995). Valued constraint satisfaction problems: Hard and easy problems. In *Proceedings of the 14th international joint conference on artificial intelligence (IJCAI-95)*, pp. 631–639.
38. Singh, S., Jaakkola, T., Littman, M. L., & Szepesv́ari, C. (2000). Convergence results for single-step on-policy reinforcement-learning algorithms. *Machine Learning, 39*, 287–308.
39. Stranders, R., Farinelli, A., Rogers, A., & Jennings, N. R. (2009). Decentralised coordination of mobile sensors using the max-sum algorithm. In *Proceedings of the 21st international joint conference on artifical intelligence (IJCAI-09).*

40. Stranjak, A., Dutta, P. S., Ebden, M., Rogers, A., & Vytelingum, P. (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)*.
41. Tel, G. (2000). *Introduction to distributed algorithms*. Cambridge: Cambridge University Press.
42. van Leeuwen, P., Hesselink, H., & Rohling, J. (2002). Scheduling aircraft using constraint satisfaction. In *Electronic notes in theoretical computer science* (pp. 252–268). Pisa: Elsevier.
43. Weiss, Y. (2000). Correctness of local probability propagation in graphical models with loops. *Neural Computation, 12*(1), 1–41.
44. Yokoo M., & Hirayama, K. (1996). 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)*, pp. 401–408.
45. Young, H. P. (1998). *Individual strategy and social structure: An evolutionary theory of institutions*. Princeton: Princeton University Press.
46. Zhang, W., Wang, G., Xing, Z., & Wittenburg, L. (2003). A comparative study of distributed constraint algorithms. In V. Lesser, C. Ortiz, & M. Tambe (Eds.), *Distributed sensor networks: A multiagent systems approach* (pp. 331–338). Boston: Kluwer.
47. Zhang, W., Wang, G., Xing, Z., & Wittenburg, L. (2005). Distributed stochastic search and distributed breakout: properties, comparison and applications to constraint optimization problems in sensor networks. *Artificial Intelligence, 161*, 55–87.
48. Zhang, W., & Xing, Z. (2002). Distributed breakout vs. distributed stochastic: A comparative evaluation on scan scheduling. In *Proceedings of the AAMAS-02 workshop on distributed constraint reasoning*, pp. 192–201.