The Knowledge Engineering Review, Vol. 00:0, 1–32. © 200X, Cambridge University Press
DOI: 10.1017/S000000000000000 Printed in the United Kingdom

A unifying framework for iterative approximate best–response algorithms for distributed constraint optimisation problems[1]

Archie C. Chapman[1], Alex Rogers[1], Nicholas R. Jennings[1] and David S. Leslie[2]

[1]*School of Electronics and Computer Science, University of Southampton, Highfield, Southampton, SO17 1BJ, UK.*
E-mail: {acc,acr,nrj}@ecs.soton.ac.uk
[2]*Department of Mathematics, University of Bristol, University Walk, Bristol, BS8 1TW, UK.*
E-mail: david.leslie@bristol.ac.uk

Abstract

Distributed constraint optimisation problems (DCOPs) are important in many areas of computer science and optimisation. In a DCOP, each variable is controlled by one of many autonomous agents, who together have the joint goal of maximising a global objective function. A wide variety of techniques have been explored to solve such problems, and here we focus on one of the main families, namely iterative approximate best–response algorithms used as local search algorithms for DCOPs. We define these algorithms as those in which, at each iteration, agents communicate only the states of the variables under their control to their neighbours on the constraint graph, and that reason about their next state based on the messages received from their neighbours. These algorithms include the distributed stochastic algorithm and stochastic coordination algorithms, the maximum–gain messaging algorithms, the families of fictitious play and adaptive play algorithms, and algorithms that use regret–based heuristics. This family of algorithms is commonly employed in real world systems, as they can be used in domains where communication is difficult or costly, where it is appropriate to trade timeliness off against optimality, or where hardware limitations render complete or more computationally intensive algorithms unusable. However, until now, no overarching framework has existed for analysing this broad family of algorithms, resulting in similar and overlapping work being published independently in several different literatures. The main contribution of this paper, then, is the development of a unified analytical framework for studying such algorithms. This framework is built on our insight that when formulated as noncooperative games, DCOPs form a subset of the class of potential games. This result allows us to prove convergence properties of iterative approximate best–response algorithms developed in the computer science literature using game theoretic methods (which also shows that such algorithms can also be applied to the more general problem of finding Nash equilibria in potential games), and, conversely, also allows us to show that many game–theoretic algorithms can be used to solve DCOPs. By so doing, our framework can assist system designers by making the pros and cons of, and the synergies between, the various iterative approximate best–response DCOP algorithm components clear.

1 Introduction

In real world applications, large–scale systems are difficult to optimally configure, often because communication restrictions, organisational structures and/or complicated topologies make it difficult, costly or impossible to collect all the necessary information at a location where a solution can be

[1]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).

computed. This, in turn, motivates the use of distributed methods of optimisation in order to find the optimal configuration. In particular, in this paper, we concentrate on multi–agent systems — that is, systems in which control is distributed across a set of autonomous agents — as an important approach to distributed optimisation. Within this context, 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. Kitano et al., 1999; Chapman et al., 2009), wide–area surveillance and distributed sensor network management (e.g. Kho et al., 2009; Hayajneh and Abdallah, 2004; Heikkinen, 2006), industrial task allocation and scheduling problems (e.g. Zhang and Xing, 2002; Stranjak et al., 2008), and the management of congested air, road, rail, and information networks (e.g. van Leeuwen et al., 2002; Roughgarden, 2005).

In more detail, in a constraint *satisfaction* problem, the aim is to find a configuration of states of variables such that they satisfy a set of constraints. A constraint *optimisation* problem is then given by a utility function that aggregates the payoffs for satisfying each of a set of 'soft' constraints (or, conversely, a penalty for violating constraints) over the states of variables in the problem (Schiex et al., 1995). A *distributed* constraint optimisation problem arises when a number of independent agents each control the state of (a subset of) the variables in the system, with the joint aim of maximising the global reward for satisfying constraints. A natural way to model DCOPs, then, is as a multi–agent system.

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 provide a unifying framework for analysing 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 (Morris, 1993) and arc consistency (Cooper et al., 2007), among others (see Apt, 2003, for more examples from the broader constraint programming literature). 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:

- *Distributed complete* algorithms, by which we mean algorithms that always find a configuration of variables that maximises the global objective function (as in finite domains one always exists). This class includes ADOPT (Asynchronous Distributed OPTimization, Modi et al. (2005)), DPOP (Dynamic Programming OPtimisation, Petcu and Faltings (2005)) and APO (Asynchronous Partial Overlay, Mailler and Lesser (2006)). Due to the inherent computational complexity of DCOPs, complete algorithms always run exponential in some aspect of their operation (i.e. the number or size of messages exchanged, or the computation performed by the agents). Furthermore, distributed complete algorithms usually operate by passing complicated data structures, or run on a highly structured ordering, such as a spanning tree, and often require additional processing of the original constraint graph.
- *Local iterative message–passing* algorithms, such as max–sum (Aji and McEliece, 2000) or distributed arc consistency (Cooper et al., 2007). In these algorithms, 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.
- *Local iterative approximate best–response* algorithms, such as the distributed stochastic algorithm (Tel, 2000; Fitzpatrick and Meertens, 2003), the maximum–gain messaging algorithm (Yokoo and Hirayama, 1996; Maheswaran et al., 2005), fictitious play (Brown, 1951; Robinson, 1951), adaptive play (Young, 1993, 1998) and regret matching (Hart and Mas-Colell, 2000). In this class, agents exchange messages containing only their state, or can observe the strategies of their neighbours. In

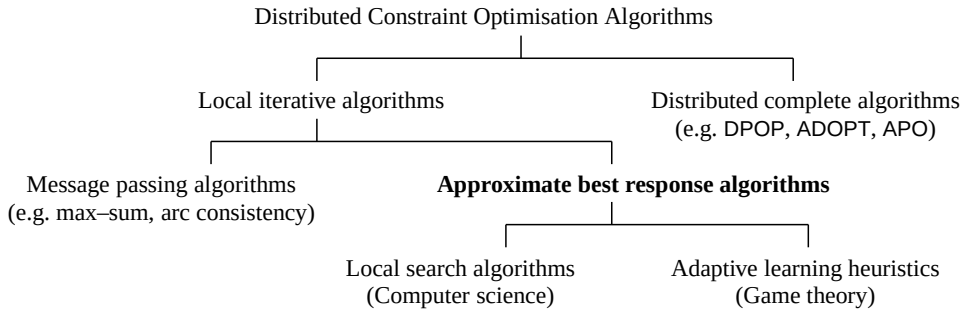


Figure 1: Taxonomy of the categories of algorithms considered in this paper, with the focus, *local iterative approximate best–response algorithms*, in bold.

game theoretic parlance, this is known as *standard monitoring*,[2] and, as the name suggests, is a typical informational assumption implicit in the literature on learning in games.

In this paper, we refer to the last two groups together as local iterative algorithms (see the taxonomy in Figure 1), to differentiate them from their complete counterparts.[3] 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 local iterative algorithms are often preferred over distributed complete algorithms.[4] 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 Krainin et al. (2007) 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. Zhang et al., 2005; Farinelli et al., 2008) and multi–robot cooperative data fusion problems (e.g. Matthews and Durrant-Whyte, 2006; Stranders et al., 2009). 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

[2]Cf. partial monitoring, as in multi–armed Bandit problems. See Blum and Mansour (2007) for a discussion of the issues surrounding these two monitoring models.

[3]Note that, when applied to DCOPs, many approaches to distributed optimisation usually can be placed into one of the three categories defined above. For example, many negotiation models and local exchange markets are, in effect, local approximate best–response algorithms. Consider a negotiation model in which at each time step one agent in each neighbourhood announces a new configuration of the variables under its control to its neighbours, with the constraint that each new configuration weakly improves the agent's payoff (as is commonly employed). This type of negotiation model is captured by our framework, as messages are local (only neighbours receive the agent's update to its state) and the process is iterative. We also point out that other approaches to the general problem of distributed optimisation that can be applied to DCOPs, such as token–passing (e.g. Xu et al., 2005) or auction protocols (e.g. Gerkey and Mataric, 2002), do not fall into any of these three categories. However, we consider an exhaustive classification and analysis of these distributed optimisation techniques outside the scope of this paper, because we are primarily concerned with local approximate best–response algorithms in the specific case of DCOPs.

[4]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 (Petcu and Faltings, 2005) or routing protocols for fixed environmental sensor networks (Kho et al., 2009).

in size, because the message size is exponential in the induced width of the communication tree (e.g. Petcu and Faltings, 2005; Rogers et al., 2009). Similarly, the number of messages exchanged in ADOPT is exponential in the height of the communication tree, and in APO, the mediator agents are required to perform computations which grow exponentially in the size of the portion of the problem they are responsible for. Such exponential relationships are simply unacceptable in embedded devices that exhibit constrained computation, 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 (Aji and McEliece, 2000) or single–looped constraint graphs (Weiss, 2000). In contrast, no such unifying and categorising framework has existed for local iterative algorithms until now. 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 (Hirayama and Yokoo, 2005) and maximum–gain messaging (Maheswaran et al., 2005) are two local approximate best–response algorithms that descend from the breakout algorithm, and distributed versions of simulated annealing have been developed for DCOPs (Fitzpatrick and Meertens, 2003; Arshad and Silaghi, 2003), 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 state. From this point of view, each agent's optimal choice is strategically dependent on the actions of its neighbouring agents (a perspective on DCOPs first adopted by Maheswaran et al., 2004), and distributed algorithms for solving such problems is the focus of the literature of learning in games (e.g. Fudenberg and Levine, 1998). However, what is common to both of these literatures is that the techniques used are all local, iterative, approximate best–response algorithms, in which agents exchange messages containing only their state, and which typically converge to local optima (or Nash equilibria); and it is game theory that has the tools and terminology to analyse algorithms that operate in such settings. 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 state (i.e. the information in the messages circulated in all of the algorithms in this class defines the appropriate solution concept).

Against this background, the main contribution of this paper is the first unifying analytical framework for studying iterative approximate best–response algorithms that are used to solve both DCOPs and potential games. Our framework is based on a problem formulation known as a *hypergraphical game* (Papadimitriou and Roughgarden, 2008) and convergence results regarding the class of *potential games* (Monderer and Shapley, 1996b). Specifically, we show that a hypergraphical game is a potential game if every local interaction can be represented as a local potential game, and this is the case for all DCOPs. We then use this framework to develop a novel parameterisation of iterative approximate best–response DCOP algorithms. In order to populate this parameterisation, we decompose algorithms proposed in both the game theory and computer science literatures in such a way so as to identify categories of substitutable components. We then analyse how these components affect the convergence properties of an algorithm employing them, using convergence analysis techniques developed specifically for potential games. As

such, our framework can be applied to potential games generally. However, due to the fact that we are considering these algorithms as distributed optimisation tools, we restrict the larger part of our discussion to the specific case of DCOP games.

In more detail, in this paper, we advance the state of the art in the following ways:

1. We derive a general result regarding hypergraphical potential games, which states that a hypergraphical game is a potential game if each of the local games is a potential game.
2. Building upon a game–theoretic formulation of DCOPs (introduced by Maheswaran et al., 2004), we show that, as a consequence of the above result, DCOP games form a subset of potential games. This allows us to apply established methods for analysing algorithms from game theory to existing algorithms produced by the computer science community, employing the Nash equilibrium condition as the relevant solution concept.
3. We develop an overarching framework that encompasses many local approximate best–response DCOP algorithms, in which we decompose the algorithms into three components: (i) a *state evaluation*, in the form of a target function; (ii) a *decision rule*, mapping from target function to a choice of state; and (iii) an *adjustment schedule*, controlling which agent updates its state when. This framework allows us to elucidate, for the first time, the relationships between the various algorithms in the form of a parameterisation of the local iterative DCOP algorithm design space. At present, various algorithms from the different disciplines use different terms for the same concepts, and are largely developed without awareness of the many similarities between them.[5]
4. By constructing such a unified view, we are able to uncover synergies that arise as a result of combining various approaches (e.g. using a particular target function and decision rule in order to reduce the communication requirement of an algorithm), and identify trade–offs in the behaviour produced by different components (e.g. choosing between adjustment schedules to produce either a slower, but anytime, algorithm, or one that converges quicker on average).

The analysis described above gives a multi–agent system designer the information needed to tailor a DCOP algorithm to their particular requirements, whether they be high quality solutions, rapid convergence, asynchronous execution or low communication costs. Moreover, such a unified approach to analysing local approximate best–response DCOP algorithms is valuable in itself, because it makes the pros and cons of the various algorithm configurations clear. Now, while this is our primary motivation, a secondary motivation is that, by stating different approaches in terms of a common framework, we can reconcile the differences in terminology that exist across the various disciplines investigating DCOPs. This, we believe, is a significant hindrance to progress in this field, and one which a unified approach can start to remove. Furthermore, we believe that our framework provides an important step towards greater use of a common specification of other problems that are examined by both the computer science and game theory communities, such as multi–agent resource and task allocation problems or the management of congested networks.

The paper progresses as follows: We begin the next section by describing DCOPs. We then introduce the notation of noncooperative games, state the Nash equilibrium solution concept for games, describe hypergraphical games, and characterise the class of potential games and their associated properties. Then, as a first step in developing our framework of the local approximate best–response DCOP algorithm design space, we show that DCOPs are potential games. Using this result, in Section 3 we populate our parameterisation of the algorithm design space with components of algorithms taken from the literatures on local search for DCOPs and learning in games. In Section 4 we discuss the connections between, and overlapping features of, game-theoretic algorithms and local approximate best–response algorithms developed by computer scientists specifically for solving DCOPs. In more detail, by characterising

[5]One notable exception to this trend is Marden et al. (2009a), who illustrate the connections between potential games and *consensus problems*. In a consensus problem, a set of agents must reach consensus upon a given value (such as a meeting point). These problems may be modelled as a DCOP containing binary and unary constraints. Each binary constraint between two agents is satisfied when their variables are set to the same value, and violations are penalised in proportion to their distance between their variables' values. However, an agent's strategy set may be limited by its unary constraints such that it is not possible for it to simultaneously satisfy all its binary constraints.

DCOPs as potential games, convergence to Nash equilibrium of the game-theoretic algorithms considered here is guaranteed. Moreover, by drawing correspondences between the game-theoretic algorithms and those developed specifically for DCOPs, these guarantees may be applied to the convergence of DCOP algorithms. Finally, Section 5 summarises our findings and discusses directions for future work.

2 DCOPs as Potential Games

In this section we show that DCOPs, when viewed from a game theory perspective (as introduced by Maheswaran et al., 2004), form a subset of the class of potential games, a useful class of games with several properties desirable to the designer of a multi–agent system. Given this insight, we can bring together the two sets of algorithms — taken from game–theory and computer science — and analyse them under a single framework, using results regarding the class of potential games. To this end, we begin this section with an overview of DCOPs. We then introduce noncooperative games and the hypergraphical game representation, in which a large game is reduced to a bipartite graph composed of agents connected to smaller *local games*. In the specific context of DCOPs, these local games correspond to constraints. We then focus on potential games, and in particular, we show that a necessary and sufficient condition for a hypergraphical game to be a potential game is that each of its local games are potential games. Now, the natural way to express constraints in a DCOP is as *team games*, which are a specific type of potential game. An important consequence of this is that DCOP games form a subset of potential games. Thus, game theoretic techniques used to analyse algorithms in potential games can be used to analyse DCOP algorithms. This insight is used in subsequent sections, where components of our algorithm parameterisation are analysed using such techniques.

2.1 Constraint Optimisation Problems

A constraint optimisation problem is represented by a set of variables $V = \{v_1, v_2, \ldots\}$, each of which may take one of a finite number of states or values, $s_j \in S_j$, a set of constraints $C = \{c_1, c_2, \ldots\}$, and a global utility function, u_g, that specifies preferences over configurations of states of variables in the system. A constraint $c = \langle V_c, R_c \rangle$ is defined over a set of variables $V_c \subset V$ and a relation between those variables, R_c, which is a subset of the Cartesian product of the domains of each variable involved in the constraint, $\prod_{v_j \in V_c} S_j$. A simple example is a binary constraint of the type typically invoked in graph colouring problems, where the relation between the two variables involved, j and k, is given by the rule that if $v_j = s$ then $v_k \neq s$. A function that specifies the reward for satisfying, or penalty for violating, a constraint is written $u_{c_k}(s_{c_k})$, where s_{c_k} is the configuration of states of the variables V_{c_k}.

Using this, the global utility function aggregating the utilities from satisfying or violating constraints takes the form:

$$u_g(s) = u_{c_1}(s_{c_1}) \oplus \ldots \oplus u_{c_k}(s_{c_k}) \oplus \ldots \oplus u_{c_l}(s_{c_l}),$$

where $\oplus$ is a commutative and associative binary operator. Now, as we are trying to generate a preference ordering over outcomes, we would like to ensure that an increase in the number of satisfied constraints results in an increase in the global utility. That is, the aggregation operator should be strictly monotonic: for any values a, b, c, if $a < b$ then $c \oplus a < c \oplus b$. Consequently, the choice of operator affects the range of values that the u_i can take. For example, if $\oplus$ is multiplication, the values of u_i must be elements of $\mathbb{R}^+$, or if $\oplus$ is addition, the values of u_i may be elements of $\mathbb{R}$. Either approach will generate a suitable ordering, however, from here on we will take the common approach of using additive aggregation functions:

$$u_g(s) = \sum_{c_k \in C} u_{c_k}(s). \tag{1}$$

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 (Schiex et al., 1995). The objective is then to find a global configuration of variable states, s^*, such that:

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

It is also possible to include hard constraints in this formalisation of DCOPs. This is achieved by augmenting the additive global utility function with a multiplicative element that captures the hard constraints:

$$u_g(s) = \prod_{hc_k \in HC} u_{hc_k}(s) \left(\sum_{sc_k \in SC} u_{sc_k}(s) \right),$$

where *HC* and *SC* are the set of hard and soft constraints, respectively, and where the payoff for satisfying each hard constraint is 1, and 0 if the hard constraint is violated. The consequence is that if any of the hard constraints are violated the global utility is 0, while if all of the hard constraints are satisfied the global utility increases with the number of soft constraints satisfied. One downside to including hard constraints in this manner is that the strict monotonicity of $u_g(s)$ is lost, meaning that a change in a variable's state may satisfy additional constraints, but not increase the global utility. If this is the case, it implies that the global utility possesses many local, sub–optimal stable points that are only quasi–local maxima.

2.2 Distributed Constraint Optimisation Problems

A DCOP is produced when a set of autonomous agents each independently control the state of a subset of the variables of a constraint optimisation problem, but share the goal of maximising the rewards for satisfying constraints (i.e. they aim to jointly maximise $u_g(s)$). For pedagogical value, and without loss of generality, we consider the case where each agent controls only one variable. We notate the set of agents involved in a constraint by N_c, and the set of constraints in which i is involved is C_i. Each agent has a *private utility function*, $u_i(s)$, which is dependent on both its own state and the state of all agents that are linked to any constraint $c \in C_i$. We call these agents i's *neighbours*, notated $\nu(i)$, and notate those neighbours involved in a specific constraint c_k as $\nu^{c_k}(i)$. The simplest and, arguably, most natural, choice of utility function in DCOP is to set each agent i's utility to the sum of the payoffs for constraints that it is involved in:

$$u_i(s) = \sum_{c_k \in C_i} u_{c_k}(s_i, s_{\nu^{c_k}(i)}). \tag{2}$$

Now, each agent's choice of strategy is guided by its desire to maximise its private utility, but this utility is strategically dependent on the strategies of its neighbours. In order to analyse such a system, we use the tools and terminology of noncooperative game theory.

2.3 Noncooperative Games

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

In noncooperative games, it is assumed that an agent's goal is to maximise its own payoff, conditional on the choices of its opponents. A *best–response correspondence*, $\beta_i(s_{-i})$, is the set of agent i's optimal strategies, given the strategy profile of its opponents, $\beta_i(s_{-i}) = \text{argmax}_{s_i \in S_i}\{u_i(s_i, s_{-i})\}$. Stable points in such a system are characterised by the set of Nash equilibria.

Definition 1 *A joint–strategy profile, s^*, such that no individual agent has an incentive to change to a different strategy, is a* ***Nash equilibrium****:*

$$u_i(s_i^*, s_{-i}^*) - u_i(s_i, s_{-i}^*) \geq 0 \quad \forall\, s_i,\ \forall\, i. \tag{3}$$

In a Nash equilibrium, each agent plays a best response: $s_i^* \in \beta_i(s_{-i}^*)$ for all $i \in N$. As such, in a game where agents independently choose which strategy to adopt, a Nash equilibrium is a stable point where no individual agent has an incentive to change their strategy.

We can also define a *strict* Nash equilibrium, which is a necessary component of many proofs of convergence in game theory, by replacing the inequality in Equation 3 with a strict inequality. The implication of this substitution is that in a strict Nash equilibrium, no agent is indifferent between their equilibrium strategy and another strategy, which is not the case in a Nash equilibrium. This also leads us to a definition of non–degenerate games. In general, a non–degenerate game is one for which in every mixed–strategy Nash equilibrium, all agents mix over the same number of pure strategies. When considering pure–strategy Nash equilibria, non–degeneracy means that for any pure–strategy equilibrium profile of its opponents, an agent's best–response correspondence contains only one strategy. Consequently, all pure–strategy Nash equilibria in non–degenerate games are strict. Note that this condition does not exclude the possibility of a game possessing multiple Nash equilibria. Rather, it ensures that at most one equilibrium exists for each of an agent's pure strategies.

2.4 Hypergraphical Games

In DCOPs, an agent's utility is a function of the constraints in which it is involved, and is only dependent on its own and its neighbours' states: i.e. $u_i(s_i, s_{\nu(i)})$. Therefore, we can model a DCOP game using a compact representation known as a *hypergraphical game*. In more detail, hypergraphical games are a model used to represent noncooperative games that have strict independences between players' utility functions (Papadimitriou and Roughgarden, 2008). In this model, the independences in the agents' utility functions are used to decompose an *n*–player global game into local games, each involving fewer players. This decomposition can be thought of as a bipartite graph, in which one set of nodes corresponds to the set of players and the other represents the games played between them. Then, any agent i whose strategy affects others players' payoffs in a particular local game is connected to that local game node. This representation is more compact than the standard normal form whenever the global game can be factored into sufficiently many local games, and when the maximum number of neighbours an agent has, k, can be bounded $k << n$, it is exponentially smaller than the standard form (Papadimitriou and Roughgarden, 2008).

Formally, a hypergraphical game comprises a set of local games: $\Gamma = \{\gamma_1, \gamma_2, ..., \gamma_m\}$. Each local game is a tuple $\gamma = \langle N^\gamma, \{S_i, u_i^\gamma\}_{i \in N^\gamma} \rangle$, where $N^\gamma \subseteq N$ is the set of agents playing γ and $u_i^\gamma : \cup_{i \in N^\gamma} S_i \rightarrow \mathbb{R}$ is the payoff to i from its involvement in γ. For each player, S_i is identical for each game, and player i chooses one strategy to play in all of the local games it is involved in (i.e. i plays the same strategy in each local game). As in DCOPs, agent i's *neighbours*, $\nu(i)$, are the agents with whom agent i shares a local game node, with those neighbours involved in a specific local game γ notated $\nu^\gamma(i)$. Agents are usually involved in more than one local game, with the set of local games in which i is involved denoted Γ_i. Agent i's total payoff for each strategy is given by the sum of payoffs from each local game it is involved in:

$$u_i(s_i, s_{-i}) = \sum_{\gamma \in \Gamma_i} u_i^\gamma(s_i, s_{\nu^\gamma(i)}).$$

In the context of DCOPs, each constraint is modelled by a local game, and agents are linked to the local games corresponding to their constraints. Each agent's utility then is given by the sum of the utilities from constraints that it is involved in (as in Equation 2).

More generally, the hypergraphical game model generalises the model of *factor graphs* (Kschischang et al., 2001). Factor graphs can be used to represent DCOPs, as well as graphical probability models such as Bayesian networks and Markov random fields. In both the factor graph and hypergraphical game models, each variable node comes under the control of an agent, and, typically, the global utility to be optimised is the sum or product of each agent's utility.[6] The difference between the models lies in what the hyperedges represent. In factor graphs, a hyperedge (factor node) represents a single valued

[6]Generally, any optimisation problem that forms a *commutative semi–ring* can be expressed as a bipartite factor graph — see Aji and McEliece (2000) for details.

function, which is its contribution to the utility of the agents it contains. In contrast, each hyperedge in a hypergraphical game represents an arbitrary noncooperative game, in which agents' payoffs may differ. In other words, a factor graph is a special case of a hypergraphical game in which each local game gives an identical payoff to all the agents involved:[7] That is, local games in a DCOP are *team games*, which are a subclass of potential games.

2.5 Potential Games

The class of potential games is characterised as those games that admit a function specifying the participant's joint preference over outcomes (Monderer and Shapley, 1996b). 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 players utility is matched by the change in the potential function. A potential function has a natural interpretation as representing opportunities for improvement to a player defecting from any given strategy profile. As the potential function incorporates the strategic possibilities of all players simultaneously, the local optima of the potential function are Nash equilibria of the game; that is, the potential function is maximised by self-interested agents in a system. Importantly, we will show that the global utility function acts as a potential for a DCOP game. We now formalise some of the key concepts related to potential games.

Definition 2 (Potential Games) *A function $P : S \rightarrow \mathbb{R}$ is a **potential** for a game if:*

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

*A game is called a **potential game** if it admits a potential.*

Intuitively, a potential is a function of action profiles such that the difference induced by a unilateral deviation equals the change in the deviator's payoff.

The usefulness of potential games lies in the fact that the existence of a potential function for a game implies a strict joint preference ordering over game outcomes. This, in turn, ensures that the game possesses a number of particularly desirable properties, which we will use to analyse the behaviour of various algorithms in the coming sections.

Theorem 3 (Monderer and Shapley (1996b)) *Every finite potential game possesses at least one pure strategy equilibrium.*

Proof Let P be a potential for a game Γ. Then the equilibrium set of Γ corresponds to the set of local maxima of P. That is, s is an equilibrium point for Γ if and only if for every $i \in N$,

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

Consequently, if P admits a maximal value in S (which is true by definition for a finite S), then P possesses a pure–strategy Nash equilibrium. □

Now, pure–strategy Nash equilibria are particularly desirable in decentralised agent-based systems, as they imply a stable, unique outcome. Additionally, strict Nash equilibria must be pure by definition. Mixed strategy equilibria, on the other hand, imply a stationary, probabilistically variable equilibrium strategy profile. Also note that it is likely that more than one Nash equilibrium exists, and that some of those Nash equilibria will be sub–optimal.

Building on this, a *step* in a game Γ is a change in one player's strategy. An *improvement step* in Γ is a change in one player's strategy such that its utility is improved. A *path* in Γ is a sequence of steps, $\phi = (s^0, s^1, \ldots, s^t \ldots)$, in which exactly one player changes their strategy at each step t. A path has an *initial point*, s^0, and if it is of finite length T, a *terminal point* s^T. A path ϕ is an *improvement path* in Γ if for

[7] Indeed, this result holds for any problem that can be represented as a factor graph and in which the variable domains are finite.

all t, $u_i(s^{t-1}) < u_i(s^t)$ for the deviating player i at step t. A game Γ is said to have the *finite improvement property* if every improvement path is finite.

Theorem 4 (Monderer and Shapley (1996b)) *Every improvement path in an ordinal potential game is finite.*

Proof For every improvement path $\phi = (s^0, s^1, s^2, \ldots)$ we have, by Equation 2:

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

Then, as S is a finite set, the sequence ϕ must be finite. □

The finite improvement property ensures that the behaviour of agents who play 'better-responses' in each period of the repeated game converges to a Nash equilibrium in finite time. Taken together, these properties ensure that a number of simple adaptive processes converge to a pure–strategy Nash equilibrium in the game (as discussed further for specific algorithms in Section 4).

Using the definitions above, we can construct a mapping between potential and hypergraphical games. To begin with, we note that Young (1998) shows that if every pairwise utility dependency corresponds to a bimatrix potential game between two agents, then the entire game is a potential game. Building on this, we generalise Young's result to hypergraphical potential games comprising several n–player games, a result upon which the rest of the paper hinges.

Theorem 5 *A hypergraphical game Γ is a potential game if every local game γ is a potential game.*

Proof Sufficiency is shown by constructing a potential function for the hypergraphical game Γ. Each local game γ has a potential $P^\gamma(s)$, so a potential for the entire game, P, can be constructed by aggregating the potentials of the local games:

$$P(s) = \sum_{\gamma \in \Gamma} P^\gamma(s).$$

Now, given a strategy profile s, a change in a deviating player i's payoff is captured by changes in the values of potential functions, $P^\gamma(s)$, of the local games i is involved in, Γ_i, so the following statements hold:

$$\begin{aligned} u_i(s_i, s_{\nu(i)}) - u_i(s'_i, s_{\nu(i)}) &= \sum_{\gamma \in \Gamma_i} u_i^\gamma(s_i, s_{\nu^\gamma(i)}) - \sum_{\gamma \in \Gamma_i} u_i^\gamma(s'_i, s_{\nu^\gamma(i)}) \\ &= \sum_{\gamma \in \Gamma} P^\gamma(s_i, s_{-i}) - \sum_{\gamma \in \Gamma} P^\gamma(s'_i, s_{-i}) \\ &= P(s_i, s_{-i}) - P(s'_i, s_{-i}), \end{aligned} \tag{4}$$

where the second line flows from the first because the potential function between independent agents is a constant value, and the third line flows from the second by definition. □

Therefore, if every local game has a potential, the global hypergraphical game also has a potential. In Section 2.6 we will use a specific instance of Theorem 5 to show that DCOP games are potential games. This result uses the fact that *team games* are a subclass of potential games. Formally, a team game is a game in which all agents share a common payoff function, and this common payoff function is a potential for the game: That is, $u_i^\gamma(s^\gamma) = P^\gamma(s^\gamma)$, $\forall\, i \in N$.

2.6 DCOPs as Graphical Potential Games

In Equation 2, we defined agents' utilities such that all agents involved in a constraint receive the same reward from that constraint; in other words, each constraint game is a team game. Consequently, we can make the following remark, which is a corollary of Theorem 5.

Corollary 6 *Every DCOP game in which the agents' private utilities are the sum of their constraint utilities is a potential game.*

As per Theorem 5, a potential for such a DCOP game can be constructed by aggregating the local (team) games' potential functions. Of course, this is exactly the global utility function u_g, specified in Equation 1. Now, for completeness, observe that, because a change in i's strategy only affects the neighbours of i, $\nu(i)$, the following statements hold:

$$\begin{aligned} u_i(s_i, s_{\nu(i)}) - u_i(s'_i, s_{\nu(i)}) &= \sum_{c_k \in C_i} u_{c_k}(s_i, s_{\nu(i)}) - \sum_{c_k \in C_i} u_{c_k}(s'_i, s_{\nu(i)}) \\ &= \sum_{c_k \in C} u_{c_k}(s_i, s_{-i}) - \sum_{c_k \in C} u_{c_k}(s'_i, s_{-i}) \\ &= u_g(s_i, s_{-i}) - u_g(s'_i, s_{-i}). \end{aligned} \tag{5}$$

Thus, any change in state that increases an agent's private utility also increases the global utility of the system.[8]

Now, when the scenario requires employing a local approximate best–response algorithm, the solution to a DCOP game is produced by the independent actions of the agents in the system. These solutions are located at stable points in the game; that is, for this class of algorithms, the Nash equilibria of the DCOP game characterise the set of solutions to the constraint optimisation problem. We have shown that DCOP games are potential games, so we are assured that at least one pure–strategy Nash equilibrium exists. Furthermore, the globally optimal strategy profile corresponds to a pure–strategy Nash equilibrium, because it is a maximum of the potential. We emphasise that in most DCOPs many Nash equilibria exist, and furthermore, many of those will be sub–optimal. This is particularly the case when hard constraints are incorporated, because, as noted earlier, the global utility function is likely to have many local quasi–maxima wherever a hard constraint is violated.

Recently, however, quality guarantees on worst–case Nash equilibria and k–optima for DCOP games have been derived for certain classes of DCOP games (Pearce and Tambe, 2007; Bowring et al., 2008). In more detail, k–optima are a generalisation of Nash equilibria, applicable only to DCOPs, that are stable in the face of deviations of all teams of size k and less. The payoff to a team, χ is defined as:

$$u_\chi(s_\chi, s_{-\chi}) = \sum_{c_k \in C_\chi} u_{c_k}(s_\chi, s_{-\chi}),$$

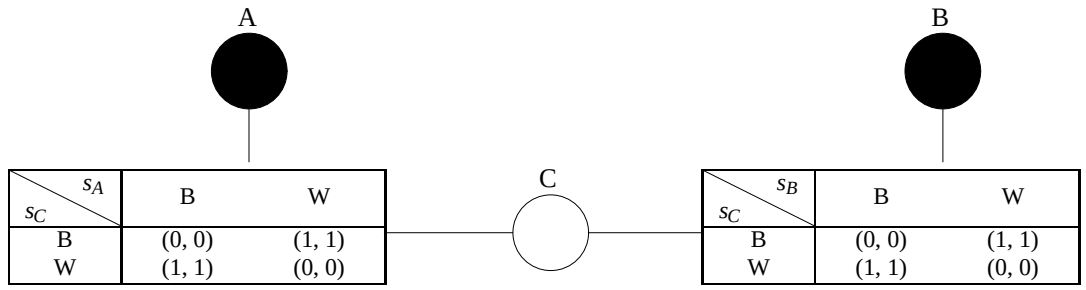
where $C_\chi = \cup_{i \in \chi} C_i$; that is, the team utility is the sum of all constraint utilities any member of χ is involved in, counting each constraint only once. Nash equilibria are k–optima with $k = 1$, and every $(k+1)$–optimum is also a k–optimum, so every k–optima is a Nash equilibrium.[9] The key result is that the worst–case k–optimal solution improves as the value of k increases (that is, as the maximum size of the deviating coalitions considered increases). Given this, the worst–case k–optimum results can be used to guarantee lower bounds on the solutions produced by some DCOP algorithms. However, bounds only exist for k greater than or equal to the constraint arity of the problem, so bounds do not exist for Nash equilibria in problems involving anything other than unary constraints.

We now present an example DCOP — a graph colouring game. We give this example because it is often used as a canonical example in this domain, and it shows that complicated payoff structures may be constructed by combining simple constraint games.

[8] Tumer and Wolpert (2004) provide another method for showing that DCOP games are potential games. Generally, any global utility function whose variables are controlled by independent agents can be instantiated as a potential game by setting each agent's utility function equal to its marginal effect on the global utility (as in the alignment criterion above). Then, by definition, any change in an agent's utility is matched by an equivalent change in the global utility. In a DCOP, one way to achieve this is to set each agent's utility to the sum of the payoffs for constraints that it is involved in, as in Equation 2.

[9] Although Nash equilibria correspond to 1–optima, note that *strong equilibria* (Aumann, 1959) do not correspond to n–optima. A strong equilibrium is robust to deviations from all coalitions of n players and less, where a coalition deviates if at least one member's individual payoff improves and none decrease. In a DCOP game, a n–optimum always exists, while a strong equilibrium may not. However, if a strong equilibrium does exist, it is n–optimal.

Example 1 *In graph colouring, neighbouring nodes share constraints, which are satisfied if the nodes are in differing states. Consider the following graph colouring problem, where each node can be either black or white, that is,* $S_i = \{B, W\}$*, and the associated* 2×2 *constraint 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 constraint games it is playing with each neighbour. In the tables below, A and B are column players and C is the row player. The top table contains the payoffs (u_A, u_B, u_C)*, and a potential for the game is given in the lower table:*

s_C \ s_A, s_B	B, B	B, W	W, B	W, W
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)

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

In the above we have described the utility functions that define a DCOP game, the associated solution concepts, and some important properties of DCOP games that flow from their classification as potential games. However, what is not specified in the above formulation are the processes by which agents adjust their states in order to arrive at a solution. These are the algorithms used to solve the game, and are the topic of the next section. Before continuing, however, we make one general comment regarding both the interpretation of the repeated game and the strategy adaptation process. We interpret the agents as being involved in several rounds of negotiation about the joint state of their variables before playing the DCOP game once. Alternatively, we could justify this perspective by assuming that the agents suffer from extreme myopia, so that they do not look beyond the immediate rewards for taking an action, as is standard in much of the literature on learning in repeated games. Either way, the only Nash equilibria that are supported are the Nash equilibria of the one-shot DCOP game; that is, the Folk Theorem for repeated games does not come into consideration.

3 A Framework for Local Approximate Best Response DCOP Algorithms

In this section we describe the basic components of the main DCOP algorithms present in the literature, including those developed for solving potential games, those developed for DCOPs generally, and those designed to solve specific problems that may be represented by a DCOP. By doing this, we open a way to investigate synergies that arise by combining seemingly disparate approaches to solving DCOPs using local approximate best–response techniques. This is pursued further in the next section, where we analyse how differences in the algorithms affect their behaviour and the solutions they produce, and propose novel algorithms incorporating the best features of each. However, first of all, an explication of the basic framework we use to analyse all the algorithms and the design space is provided.

As noted above, in all the DCOP algorithms we discuss, agents act myopically, in that they only consider the immediate effects of their actions (or state changes, see the discussion above). Given an appropriate trigger, the individual agents follow the same basic two–stage routine, comprising *state evaluation*, which produces some measure of the desirability of each state, followed by a *decision* on

which action to take, based on the preceding state evaluations. The system–wide process that triggers an agent's processes is given by an *adjustment schedule*, that controls which agent adjusts its state at each point in time. In more detail:

State Evaluation: Each algorithm has a target function that it uses to evaluate its prospective states. The target functions are typically functions of payoffs, and sometimes take parameters that are set exogenously to the system or updated online. 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 states to decide upon an action to take. Typically an algorithm prescribes that an agent selects either the state that maximises or minimises the target function (depending on the target function in question), or selects a state probabilistically, in proportion to the value of the target function for that state.

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' states), whereas in algorithms that use preferential adjustment schedules (such as the maximum–gain messaging algorithm), additional information may be required to run the adjustment schedule.

Given this background, this section examines the forms that each of the three algorithm stages can take. In doing so, we make clear, for the first time, the many connections between the various algorithms. During this section we will be referring to many algorithms from the literature on DCOP algorithms and learning in games, the most important being:

- Best response and smooth best response (Fudenberg and Levine, 1998);
- Better–reply dynamics (Mezzetti and Friedman, 2001);
- Distributed stochastic algorithm (Tel, 2000; Fitzpatrick and Meertens, 2003);
- Maximum–gain messaging algorithm (Yokoo and Hirayama, 1996; Maheswaran et al., 2005);
- Adaptive play (Young, 1993) and spatial adaptive play (Young, 1998, Chapter 6);
- Distributed simulated annealing (Arshad and Silaghi, 2003);
- Fictitious play (Brown, 1951; Robinson, 1951) and smooth fictitious play (Fudenberg and Kreps, 1993; Fudenberg and Levine, 1998);
- Joint strategy fictitious play (Marden et al., 2009b);
- Regret matching (Hart and Mas-Colell, 2000) and variants of regret monitoring (Arslan et al., 2007);
- Stochastic coordination–2 algorithm and maximum–gain messaging–2 algorithm (Maheswaran et al., 2005);

We now discuss the various target functions that are used in DCOP algorithms, and then examine different decision rules and adjustment schedules used in DCOP algorithms.

3.1 State Evaluations

The way in which a local iterative approximate best–response algorithm searches the solution space is, in the largest part, guided by the target function used by agents to evaluate their choice of state. The most straightforward approach is to directly use the payoffs given by the utility functions to evaluate states. Some shortcomings of this approach, such as slow convergence, poor search and sub–optimal solutions,

are addressed by more sophisticated specifications of the algorithm's target function. These include using measures of expected payoff, average regret, and aggregated utilities. The next subsection addresses using immediate payoffs as a target function, while subsequent ones examine the more sophisticated target functions.

3.1.1 Immediate Payoff and Change in Payoff

As noted above, the simplest target function that a DCOP algorithm can use to evaluate its strategy is to directly use its private utility function, $u_i(s_i, s_{-i})$, producing typical 'hill-climbing' or 'greedy' behaviour. This leads the system to a Nash equilibrium, which corresponds to a local potential–maximising point. The best–response dynamics is the most well known example of such an approach.

Furthermore, many algorithms, including the distributed stochastic algorithm, the distributed breakout algorithm and the maximum–gain messaging algorithms, use the amount to be gained by changing strategy as a target function. This is a simple perturbation of the utility function achieved by finding the difference between the current state's value and the value of all other possible states. For many decision rules, using either the gain or the raw utility function as an input will produce the same result. However, when it is useful to differentiate between those states that improve payoff and those that do not, or when the decision rule used can only take non–negative values as inputs, gain in payoff is the appropriate target function.

Agents using this target function to update their evaluation of states only need to observe the current state of their neighbours to run the algorithm, and do not need to communicate any further information. However, the use of such a target function can often result in slow convergence.

3.1.2 Expected Payoff Over Historical Frequencies

In order to speed up convergence, an algorithm can use the expected payoff for each state over historical frequencies of state profiles as a target function. These can be constructed in at least two different ways, either by maintaining an infinite memory of past actions, as in the fictitious play algorithms, or a finite memory, as in variants of adaptive play.

First, we consider the infinite memory case, and the fictitious play target function in particular. Let agent j's *historical frequency* of playing s'_j, be defined as:

$$q^t_{s'_j} = \frac{1}{t}\sum_{\tau=0}^{t-1} I\{s'_j = s^\tau_j\},$$

where $I\{s'_j = s^\tau_j\}$ 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:

$$q^t_{s'_j} = \frac{1}{t}\left[I\{s'_j = s^{t-1}_j\} + (t-1)q^{t-1}_{s'_j}\right].$$

Now, $q^t_{s'_j}$ 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 as a vector of historical frequencies of play for each $j \neq i$ is:

$$q^t_j = \begin{bmatrix} q^t_{s^1_j} \\ \vdots \\ q^t_{s^{|S_j|}_j} \end{bmatrix},$$

and i's belief over all of its opponents' actions is the set of vectors $q^t_{-i} = \{q^t_j\}_{j\in N\setminus i}$. Following this, i's expected payoff for playing s'_i, given q^t_{-i}, is then:

$$FP^t_i(s'_i, q^t_{-i}) = \sum_{s_{-i}\in S_{-i}}\left[u_i(s'_i, s_{-i})\prod_{s_j\in s_{-i}} q^t_{s_j}\right] \tag{6}$$

where, in general, $S_{-i} = \cup_{j\neq i\in N} S_j$. However, note that in any hypergraphical game, such as a DCOP game, q_{-i} and S_{-i} may be replaced with $q_{\mathrm{v}(i)}$ and $S_{\mathrm{v}(i)}$, respectively.

The classical fictitious play and smooth fictitious play algorithms use this measure of expected payoff as a target function. 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 (Leslie and Collins, 2006). For example, in situations where an agent can only observe its payoff and has no knowledge of its neighbours' actions, the expected payoff may be calculated as the average received payoff to each action. This is known as *cautious* or *utility-based* fictitious play (Fudenberg and Levine, 1995), and, as noted by Arslan et al. (2007), is effectively a payoff-based reinforcement learning algorithm. Additionally, Crawford (1995) suggests a *weighted* fictitious play process for highly variable environments, in which the contribution of past observations to an agent's belief are exponentially discounted.

A similar infinite memory process, called *joint–strategy fictitious play* (JSFP), was introduced by Marden et al. (2009b). 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:

$$JSFP_i^t(s_i', q_{-i}^t) = \sum_{s_{-i} \in S_{-i}} q_i^t(s_{-i}) u_i(s_i', s_{-i}).$$

This can be expressed more simply as:

$$JSFP_i^t(s_i', q_{-i}^t) = \frac{1}{t} \sum_{\tau=1}^{t} u_i(s_i', s_{-i}^\tau),$$

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 state, rather than the full action history:

$$JSFP_i^t(s_i', q_{-i}^t) = \frac{1}{t} \left[u_i(s_i', s_{-i}^t) + (t-1) JSFP_i^{t-1}(s_i', q_{-i}^{t-1}) \right], \tag{7}$$

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.

Marden et al. (2009b) show that the classical fictitious play and the joint–strategy fictitious play target functions coincide in all two–player games. As an example of the usefulness of the hypergraphical game representation, we now show that the classical fictitious play and the joint–strategy fictitious play target functions are identical in all binary hypergraphical games (i.e. every local game is a two–player game). To begin with, in any hypergraphical game, Equation 6 can be expressed in terms of the sum of utilities from local games:

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

Now, in a binary hypergraphical game, the product term is redundant because an agent has only one opponent in each local game, so using $q_{s_j'}^t = \frac{1}{t} \sum_{\tau=0}^{t-1} I\{s_j' = s_j^\tau\}$, we can rewrite the above as:

$$FP_i^t(s_i', q_{-i}^t) = \sum_{\gamma \in \Gamma_i} \sum_{s_{j^\gamma} \in S_{j^\gamma}} u_i^\gamma(s_i', s_{j^\gamma}) \frac{1}{t} \sum_{\tau=0}^{t-1} I\{s_{j^\gamma} = s_{j^\gamma}^\tau\},$$

where j^γ is i's opponent in γ. The expression above can be simplified because the combination of summing over all S_{j^γ} and the indicator function can be replaced by taking the average of the payoffs actually received in each local game over time. This gives:

$$FP_i^t(s_i', q_{-i}^t) = \frac{1}{t} \sum_{\tau=0}^{t-1} \sum_{\gamma \in \Gamma_i} u_i^\gamma(s_i', s_{j^\gamma}^\tau),$$

which, like the joint–strategy fictitious play target function, admits a recursive specification:

$$\begin{aligned} FP_i^t(s_i', q_{-i}^t) &= \frac{1}{t}\left[\sum_{\gamma\in\Gamma_i} u_i^\gamma(s_i', s_{j^\gamma}^t) + (t-1)FP_i^{t-1}(s_i', q_{-i}^{t-1})\right] \\ &= \frac{1}{t}\left[u_i(s_i, s_{-i}^t) + (t-1)FP_i^{t-1}(s_i', q_{-i}^{t-1})\right]. \end{aligned}$$

This form of classical fictitious play for binary hypergraphical games is identical to the definition of the joint–strategy fictitious play target function stated in Equation 7.

We now consider a class of processes that evaluate the expected payoff for each state over historical frequencies of state profiles computed from samples taken from a finite memory, called adaptive play (Young, 1993). In adaptive play, agents maintain a finite history over their opponents' actions, and construct an estimate of their mixed strategies by sampling from this history. Each individual only has a finite memory, of length m, and recalls the previous m actions taken by opponents. On each play of the game, each individual takes a sample of size $k \leq m$ from this memory, and computes an estimate of its opponents' actions from this sample. That is, i's belief over j's actions, $q_{(s_j)}{}^{i,t}$, is given by the proportion of times that j has played strategy s_j in the sample of size k. Then, as in fictitious play, i's expected payoff for playing s_i', given $q_{-i}^{i,t}$, is given by:

$$AP_i^t(s_i', q_{-i}^{i,t}) = \sum_{s_{-i}\in S_{-i}} \left[u_i(s_i', s_{-i}) \prod_{s_j\in s_{-i}} q_{s_j}^{i,t}\right]. \tag{8}$$

All of the adaptive play variants use this type of state evaluation, with various constraints on the relative values of m and k. In particular, spatial adaptive play was described in Young (1998) as a variation of adaptive play in which both the memory m and the sample size k are 1, and in which only a single agent updates its strategy each time step.

The fictitious play, joint–strategy fictitious play and adaptive play target functions have the same communication requirements as algorithms that use the immediate payoff for an action as a target function, because at each point in time each agent only needs to know the values of s_{-i} in order to update its evaluation of each of its states.

3.1.3 Average Regret for Past Actions

Another approach that can be used to speed up convergence is to measure the average 'regret' for not taking an action, where the regret measure for a particular strategy at a particular time is the difference between the payoff that would have been received for playing that strategy at time τ and the strategy that was actually chosen at τ. The average of these differences over the history of repeated play is the average regret for not adopting that particular strategy consistently over the entire history of play:

$$AR_i^t = \frac{1}{t}\sum_{\tau=1}^{t}\left[u_i(s_i', s_{-i}^\tau) - u_i(s_i^\tau, s_{-i}^\tau)\right]$$

This target function is also known as *external regret*. Like the measure of expected payoff based on joint strategies discussed above, the average regret target function can be specified recursively, only requiring the agents to maintain a measure of average regret for each state:

$$AR_i^t = \frac{1}{t}\left[u_i(s_i', s_{-i}^t) - u_i(s^t) + (t-1)AR_i^{t-1}\right]. \tag{9}$$

Hart and Mas-Colell (2000) use this target function to construct their regret matching algorithm, and use it to characterise an entire class of adaptive strategies (Hart and Mas-Colell, 2001). It is also used as the target function for a distributed simulated annealing method for finding the Nash equilibria of games (La Mura and Pearson, 2002). Like fictitious play, many variants of the method of updating regrets have been suggested. For example, a variation of average regret for situations where an agent can only observe its own payoff is known as *internal regret* (Blum and Mansour, 2007). This method calculates regret as the

difference between the average payoff for choosing each state in the past and the received payoff for the state selected at a particular time. In this way it is analogous to cautious fictitious play.

Another example, proposed by Arslan et al. (2007), is a regret–based target function in which past regrets are weighted by a constant value. In other words, past regrets are exponentially discounted, or the agents have 'fading memory':

$$WR_i^t = \rho \left[u_i(s_i, s_{-i}^t) - u_i(s^t) \right] + (1-\rho) WR_i^{t-1}, \tag{10}$$

where $(1-\rho)$ is the discount factor, $0 < \rho \leq 1$.

Again, this target function uses the same observations as algorithms that use the immediate payoff, fictitious play, joint–strategy fictitious play and adaptive play target functions payoff for an action as a target function, because at each time–step, an agent only needs to know the values of s_{-i} in order to update its regret for each of its states.

3.1.4 Aggregated Immediate Payoffs

One inconvenient aspect of the above target function specifications is that they are prone to converging to suboptimal equilibria (in the absence of some ergodic process such as a random perturbation to payoffs, as will be discussed in Section 3.2). A number of algorithms avoid this problem by using aggregated payoffs to evaluate states. However, these algorithms have significantly increased communication requirements, as agents pass information regarding the value of each state, rather than just indicating their current state.

The maximum–gain messaging–2 algorithm and stochastic coordination–2 algorithm both use a pairwise aggregate of local utility functions to evaluate the joint state of any two agents, i and j:

$$u_{ij} = \sum_{c_k \in C_i} u_{c_k}(s_i, s_j, s_{-\{i,j\}}) + \sum_{c_k \in C_j} u_{c_k}(s_i, s_j, s_{-\{i,j\}}) - \sum_{c_k \in C_i \cap C_j} u_{c_k}(s_i, s_j, s_{-\{i,j\}}), \tag{11}$$

where the final term adjusts for the double counting of any constraints shared by the agents. This target function allows the agents to evaluate synchronised state changes, and can be used to avoid the worst Nash equilibria in the system by converging only to 2–optima. As mentioned in Section 2.2, Pearce and Tambe (2007) show that the worst–case 2–optimum solution to a DCOP game is greater than that for a Nash equilibrium, or 1–optimum. Thus, this result implies that an algorithm that uses a pairwise aggregated target function has a higher lower bound solution than any algorithm that only converges to a Nash equilibrium.

Furthermore, Maheswaran et al. (2005) propose two families of k–coordinated algorithms —the maximum–gain messaging–k and stochastic coordination–k algorithms — that use locally aggregated utilities for coalitions of k agents, which each converge to an element of their respective set of k–optima. However, although the number of messages communicated at each step to calculate the aggregated utilities increase linearly with the number of neighbours each agent has, the size of each message increases exponentially with the size of the coalitions. These factors make constructing algorithms that aggregate the utilities of large coalitions of agents infeasible.

3.2 Decision Rules

A decision rule is the procedure that an agent uses to map the output of its target function to its choice of strategy. The decision rule used by most DCOP algorithms is either the *argmax* or *argmin* functions, or probabilistic choice functions. The choice between these two serves an important purpose, as it determines whether the algorithm follows a hill-climbing trajectory or is stochastic. In the former case, the algorithm produced may converge quickly — or may even be anytime — but it may not be able to escape from the basin of attraction of a local maximum.

On the other hand, adding ergodicity allows the algorithm to escape from the basin of attraction of a sub-optimal Nash equilibrium (or local maximum of the potential function), but at the cost of sometimes degrading the solution quality. Proportionally probabilistic decision rules map payoffs through a probabilistic choice function to a mixed strategy (Fudenberg and Levine, 1998). As such, states with a

higher–valued target function are chosen with greater probability, but states with lower payoffs than the current state are sometimes chosen. This allows the agents in the system to escape local optima. However, it also means that the algorithm is no longer an anytime optimisation algorithm. Two such probabilistic choice functions are the *linear* and *multinomial logit* choice models. The *simulated annealing* decision rules add ergodicity by probabilistically moving to a lower utility state in proportion to its distance from the current state's utility, while always moving to states with higher utility. Finally, the ε–greedy decision rule, commonly used in machine learning, selects a state with the highest valuation with probability $(1-\varepsilon)$ and chooses uniformly from the remaining states with probability ε. We now consider these rules in more detail.

3.2.1 Argmax and Argmin Decision Rules

The *argmax* function (or, equivalently, the *argmin* function) returns the state with the highest (lowest) valued target function. Two variations of this decision rule are present in the literature, which differ in how they handle multiple situations where multiple states correspond to the highest value of the target function. These two variants of the *argmax* function are discussed in the context of the distributed stochastic algorithm in Zhang et al. (2005), where the algorithms are named DSA–A and DSA–B, respectively. In the first, which we call *argmax–A*, if the agent's state at $t-1$ is one of the states that maximises the target function, then it is the state selected at t. Otherwise, a new state is randomly chosen with equal probability from the set of target function maximising states. That is, the agent only changes its state if it improves the value of its target function. In the second variant, *argmax–B*, an agent randomly selects a new state from the set of target function maximising states, without regard for the state at $t-1$. Note that in non–degenerate games, every best response is unique, so the two variants of the argmax function behave identically.

A benefit is that using the *argmax* function in conjunction with an immediate reward target function and a suitable adjustment schedule (such as in the maximum–gain messaging algorithm) is that the resulting algorithm can be 'anytime', in that each new solution produced is an improvement on the last. However, one potential drawback of this technique is its dependence on initial conditions, even when the algorithm is not anytime. It is possible that the initial random configuration of states places the system outside the basin of attraction of an optimal Nash equilibrium, meaning that an algorithm using the *argmax* decision rule can never reach an optimal point. To avoid this scenario, a probabilistic decision rule may be used instead.

3.2.2 Linear Probabilistic Decision Rules

The linear probabilistic decision rule produces a mixed strategy with probabilities in direct proportion to the target value of each state:

$$Pr_{s_i} = \frac{u_i(s_i, s^t_{-i})}{\sum_{s_i \in S_i} u_i(s_i, s^t_{-i})}.$$

This model is only appropriate when the target function supplies a non-negative input. Although this appears to be quite a substantial limitation, the linear probabilistic choice rule is useful in certain circumstances. For example, the regret matching algorithm uses the linear probabilistic choice function with negative regrets set equal to zero, so that they are chosen with zero probability (Hart and Mas-Colell, 2000). Another example is the better–reply dynamic, in which an agent randomly chooses a new state from those which (weakly) improve its payoff (Mezzetti and Friedman, 2001).

3.2.3 Multinomial Logit Decision Rules

One probabilistic decision rule that can accept negative input is the multinomial logit decision rule (Anderson et al., 1992), known in statistical mechanics as the Boltzmann distribution:

$$Pr_{s_i}(\eta) = \frac{e^{\eta^{-1} u_i(s_i, s^t_{-i})}}{\sum_{s_i \in S_i} e^{\eta^{-1} u_i(s_i, s^t_{-i})}}. \tag{12}$$

Here states are chosen in proportion to their reward, but their relative probability is controlled by η, a temperature parameter. If $\eta = 0$ then the *argmax* function results, while $\eta = \infty$ produces a uniform distribution across strategies, which results in the state of the system following a random walk. Depending on the specifics of the problem at hand, the temperature can be kept constant or may be decreased over time. If an appropriate cooling schedule is followed, the later case is referred to as a 'greedy in the limit with infinite exploration' decision rule in the online reinforcement learning literature (Singh et al., 2000). The multinomial logit choice function is used in typical specifications of smooth best response, spatial adaptive play (Young, 1998, Chapter 6) and smooth fictitious play (Fudenberg and Levine, 1998; Hofbauer and Sandholm, 2002).

3.2.4 Simulated Annealing Decision Rules

The simulated annealing decision rule is a probabilistic decision rule that works by randomly selecting a new candidate state, k, and accepting or rejecting it based on a comparison to the current state (Metropolis et al., 1953; Kirkpatrick et al., 1983). All improvements in the target function are accepted, while states that lower the value of the target function are only accepted in proportion to their distance from the current state's value. For example, the case where the target function is given by the agent's private utility function gives the following decision rule:

$$Pr_{s_i}(\eta) = \begin{cases} 1 & \text{if } u_i(k, s_{-i}) \geq u_i(s_i, s_{-i}) \\ e^{\eta^{-1}(u_i(k,s_{-i}) - u_i(s_i,s_{-i}))} & \text{otherwise ,} \end{cases} \tag{13}$$

where $u_i(k, s_{-i})$ and $u_i(s_i, s_{-i})$ are the candidate and the current state's payoffs, respectively. As with the multinomial logit choice model (Equation 12), η is a temperature parameter. If $\eta = 0$ then only states that improve the target function are accepted, while $\eta = \infty$ means that all candidate states are accepted, and consequently, as with the multinomial logit function, the state of the system follows a random walk. The temperature may be kept constant, resulting in an analogue of the Metropolis algorithm (Metropolis et al., 1953), or may be decreased over time as in a standard simulated annealing optimisation algorithm (Kirkpatrick et al., 1983). Distributed simulated annealing has been proposed as a global optimisation technique for DCOPs (Arshad and Silaghi, 2003), and a simulated annealing algorithm based on average regret has been suggested as a computational technique for solving the Nash equilibria of general games (La Mura and Pearson, 2002).

3.2.5 The ε–Greedy Decision Rule

One particularly common decision rule used in online reinforcement learning is known as ε–greedy. Under this rule, an agent selects a state with maximal expected reward with probability $(1 - \varepsilon)$, and a random other action with probability ε, i.e.:

$$Pr_{s_i}(\varepsilon) = \begin{cases} 1 - \varepsilon & \text{if } s_i = \underset{k \in S_i}{\operatorname{argmax}} \left[u_i(k, s_{-i}) \right] \\ \varepsilon & \text{otherwise ,} \end{cases} \tag{14}$$

Like the multinomial logit decision rule, the exploration parameter, ε, can be kept constant or may be decreased over time. Under specific conditions on the rate of decrease, the later case is another example of a 'greedy in the limit with infinite exploration' decision rule (Singh et al., 2000). This decision rule is used in many variations of adaptive play (e.g. Young, 1993).

3.3 Adjustment Schedules

An adjustment schedule is the mechanism that controls which agents adjust their state at each point in time. The simplest schedule is the 'flood' schedule, where all agents adjust their strategies at the same time. Beyond this, adjustment schedules can be divided into two groups: random or deterministic. The former are typically run by each agent independently, and can produce sequential or parallel actions by agents. The latter often require agents to communicate information between themselves in order to

coordinate which agent adjusts its strategy at a given point in time, with priority usually given to agents that can achieve greater gains or are involved in more conflicts. Other times the ordering is decided upon in a preprocessing stage of the algorithm.

3.3.1 Flood Schedule

Under the flood schedule, all agents adjust their strategies at the same time. This schedule is in essence the Jacobi iteration method (Press et al., 1992). It is frequently used in applications of local greedy algorithms (e.g. Matthews and Durrant-Whyte, 2006)), and in implementations of fictitious play (e,g, Leslie and Collins, 2006)) and some variants of adaptive play (e.g. Young, 1993).

A problem commonly observed with algorithms using the flood schedule, particularly greedy algorithms, is the presence of 'thrashing' or cycling behaviour (Zhang et al., 2005). Thrashing occurs when, as all agents adjust their states at the same time, they inadvertently move their joint state to a globally inferior outcome. Furthermore, it is possible that a set of agents can become stuck in a cycle of adjustments that prevents them from converging to a stable, Nash equilibrium outcome. In theory, the potential for these types of behaviours to occur means that convergence cannot be guaranteed, while in practice they are detrimental to the performance of any algorithm using the flood schedule.

3.3.2 Parallel Random Schedules and Inertia

Parallel random adjustment schedules are simply variations of the flood schedule, in which each agent has some probability p of actually changing its state at any time step. In the computer science literature on DCOPs, p is known as the 'degree of parallel executions' (Zhang et al., 2005), whereas in the game theory literature it is commonly referred to as choice 'inertia' (e.g. Mezzetti and Friedman, 2001; Marden et al., 2009b).

Now, this type of adjustment schedule does not ensure that thrashing is entirely eliminated. However by selecting an appropriate value of p, thrashing and cycling behaviour can be minimised, producing an efficient algorithm with parallel execution without increasing the communicational requirements. Furthermore, inertia is essential to the convergence proofs of various processes, such as the better–reply dynamics and joint–strategy fictitious play. This is the adjustment schedule used by the distributed stochastic algorithm, regret matching, and joint–strategy fictitious play with inertia.

3.3.3 Sequential Random Schedules

The group of adjustment schedules that we call *sequential random schedules* involve randomly giving one agent at a time the opportunity to adjust its strategy, with agents selected by some probabilistic process. The motivation for using this adjustment schedule is grounded in the convergence proofs for many of the adaptive procedures taken from the game theory literature. In particular, the finite improvement property of potential games directly implies that agents that play a sequence of 'better responses' converge to a Nash equilibrium in a finite number of steps. This property is used to prove the convergence of spatial adaptive play and a version of Fictitious Play with sequential updating (Berger, 2007).

Now, sequential procedures do not allow for the parallel execution of algorithms in independent subgraphs, where thrashing is not a concern, or for the execution of algorithms whose convergence can be guaranteed without asynchronous moves. However, they do ensure that agents do not cycle or thrash, which is a risk with using the flood or parallel random adjustment schedules.

In practice, there are a number of ways to implement this type of schedule. A particularly straightforward approach which ensures that all agents have an opportunity to adjust their state is given by dividing time into segments, with each agent randomly selecting a point from a uniform distribution over the segment at which to adjust its strategy. Agents then adjust their states sequentially and in a random order, which satisfies the assumptions of the theoretical convergence results. However, such a schedule may depend on an external or synchronised clock. This type of schedule is essentially a form of Gauss-Seidel iteration, in which the order of updating is shuffled each cycle (Press et al., 1992). We refer to this schedule as a shuffled sequential schedule. Another simple approach would be to give each agent a mechanism that triggers action according to a probabilistic function of time, such as an exponential distribution, which

can be run on an internal clock. In this process the probability of any two agents adjusting their state at the same time is zero. This could be called a sequential random exponential schedule. A final suggestion is to use token passing to maintain sequential updates.

3.3.4 Maximum–Gain Priority Adjustment Schedule

The maximum–gain messaging algorithm takes its name from the type of adjustment schedule it uses (Maheswaran et al., 2005; Yokoo and Hirayama, 1996). This preferential adjustment protocol involves agents exchanging messages regarding the maximum gain they can achieve. If an agent can achieve the greatest gain out of all its neighbours, then it implements that change, otherwise it maintains its current state. The maximum–gain messaging adjustment schedule avoids thrashing or cycling, as no two neighbouring agents will ever move at the same time.

3.3.5 Constraint Priority Adjustment Schedule

A second preferential adjustment schedule, the constraint priority adjustment schedule, works by allocating each agent a priority measure based on the number of violated constraints it is involved with. This is the type of adjustment schedule used by the APO algorithm.

4 Local Approximate Best Response Algorithm Parametrisation

In this section we discuss how the different components of a DCOP algorithm, as identified in Section 3, affect the quality and timeliness of the solutions it produces. As an overview, Table 1 presents the parameterisation of the main local approximate best–response DCOP algorithms highlighted in Section 3: two versions of the distributed stochastic algorithm (DSA–A and DSA–B), the maximum–gain messaging algorithm (MGM), the better–reply dynamic with inertia (BR–I), spatial adaptive play (SAP), distributed simulated annealing (DSAN), fictitious play (FP) and smooth fictitious play (smFP), joint–strategy fictitious play with inertia (JSFP–I), adaptive play (AP), regret matching (RM), weighted regret monitoring with inertia (WRM–I), the stochastic coordination–2 algorithm (SCA–2), and the maximum–gain messaging–2 algorithm (MGM–2). In this table, the relationships between the algorithms are clearly shown, in terms of the components used to construct each of them.

Before beginning the detailed discussion, we define some of the terms we use in the analysis. In particular, we say an algorithm *converges in finite time* if there exists a value T after which the joint state of the agents is guaranteed to be a Nash equilibrium. An algorithm *almost surely converges* if the probability of the agents' joint state 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 states converges to some specified distribution as time tends to infinity. Typically, the specified distribution is the Boltzmann distribution over the joint state 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 (Grimmett and Stirzaker, 2001). 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 Grimmett and Stirzaker, 2001, for example).

We now move on to discuss in detail how the algorithms are related and where they differ, and furthermore, how this affects their behaviour and the solutions they produce. In the process, we will sketch several convergence proof techniques, and will extend some existing convergence proofs to cover algorithms with similar structures. First, we analyse those algorithms that use immediate reward or change in payoff as a target function. This allows us to demonstrate clearly how different decision rules and adjustment schedules affect the convergence properties of the algorithms. We then discuss algorithms that use recursive averaging measures, such as expected reward or regret, as target functions.

	Target Function	Memory	Decision Rule	Adjustment Schedule
DSA–A	$u_i(s_i, s_{-i})$	—	argmax–A	Parallel random (p)
DSA–B	$u_i(s_i, s_{-i})$	—	argmax–B	Parallel random (p)
MGM	$u_i(s_i, s_{-i})$	—	argmax–B	Preferential: Maximum gain
BR–I	$u_i(s_i, s_{-i})$	—	argmax–B	Flood
SAP	$u_i(s_i, s_{-i})$	—	Logistic (η)	Sequential random
DSAN	$u_i(s_i, s_{-i})$	—	Sim. annealing (η)	Parallel random (p)
FP	$\sum_{s_{-i} \in S_{-i}} \left[u_i(s'_i, s_{-i}) \prod_{s_j \in s_{-i}} q^t_{s_j} \right]$	Opponents' freq. of play	argmax–B	Flood
smFP	$\sum_{s_{-i} \in S_{-i}} \left[u_i(s'_i, s_{-i}) \prod_{s_j \in s_{-i}} q^t_{s_j} \right]$	Opponents' freq. of play	Logistic (η)	Flood
JSFP–I	$\frac{1}{t} \left[\begin{array}{c} u_i(s_i, s^t_{-i}) \\ +(t-1) JSFP_i^{t-1} \end{array} \right]$	Average expected utility	argmax–A	Parallel random (p)
RM	$\frac{1}{t} \left[\begin{array}{c} u_i(s_i, s^t_{-i}) - u_i(s^t) \\ +(t-1) AR_i^{t-1} \end{array} \right]$	Average regrets	Linear prob$^+$	Parallel random (p)
WRM–I	$\rho \left[u_i(s_i, s^t_{-i}) - u_i(s^t) \right] + (1-\rho) WR_i^{t-1}$	Discounted average regrets	Linear prob$^+$ *or* logistic$^+$ (η)	Parallel random (p)
SCA–2	$u_i(s_i, s_{-i}) + u_j(s_j, s_{-j}) - \sum_{c_k \in C_i \cap C_j} u_{c_k}(s_i, s_j, s_{-\{i,j\}})$	—	argmax–A	Parallel random (p)
MGM–2	$u_i(s_i, s_{-i}) + u_j(s_j, s_{-j}) - \sum_{c_k \in C_i \cap C_j} u_{c_k}(s_i, s_j, s_{-\{i,j\}})$	—	argmax–B	Preferential: Maximum gain

Table 1 Parameterisation of the main local approximate best–response DCOP algorithms.

4.1 Immediate Reward and Gain–Based Algorithms

In this section we discuss those algorithms that use immediate payoff, change in payoff, or aggregated measures of either to evaluate states. In so doing, we will demonstrate three techniques of proving the convergence of a DCOP algorithm, which exploit the existence of a potential function in three different ways. Importantly, these techniques can be extended to similar algorithms that comprise common components. As such, we discuss the algorithms in groups based on their convergence proofs, beginning with MGM which has the anytime property and converges to Nash equilibrium. Second, we consider DSA and BR–I, which rely on almost–sure convergence to Nash equilibrium. Third, we discuss SAP and DSAN, which, by virtue of the particular probabilistic decision rules they employ, can be shown to converge in distribution to the global maximum of the potential function. Finally, we discuss the convergence of the MGM–k and SCA–k to k–optima, and relate their convergence to that of MGM and DSA, respectively

4.1.1 Anytime Convergence of MGM

We begin with MGM. In ordinal potential games, MGM converges to a Nash equilibrium and is an anytime algorithm (Maheswaran et al., 2005). This is because agents act in isolation (i.e. none of their neighbours change strategy at the same time), so their actions only ever improve their utility, which implies an improvement in global utility (by Equation 5). Furthermore, by the same reasoning, the finite improvement property ensures that this algorithm converges to a Nash equilibrium in finite time.[10]

[10] Maheswaran et al. (2005) show that MGM is anytime and converges to an element in the set of Nash equilibrium in DCOP games directly, without using a potential game characterisation of the problem.

4.1.2 Almost–Sure Convergence of DSA and BRI

Although similar in construction to MGM, neither DSA nor BR–I are anytime, as it is possible that agents who change state at the same time find themselves in a worse global state than they began in. However, using almost–sure convergence, we can show the following: DSA–A (DSA using *argmax*–A) almost surely converges to a Nash equilibrium, and DSA–B and BR–I almost surely converge to a strict Nash equilibrium. Although similar results have been published (e.g. Young, 1998; Mezzetti and Friedman, 2001), for pedagogical value, we present a proof of the convergence of DSA–B to a strict Nash equilibrium, which we use as a template for sketching other convergence proofs. We will refer back to the steps presented in this proof when discussing the convergence of other algorithms in future sections.

Proposition 7 *If a strict Nash equilibrium exists, then DSA–B almost surely converges to a strict Nash equilibrium in repeated potential games.*

Proof A strict Nash equilibrium is an absorbing strategy profile under DSA–B's dynamics; that is, once in a strict Nash equilibrium, no agent will change their strategy. Now, for any non–Nash equilibrium outcome, there exists a *minimal improvement path*, terminating at a Nash equilibrium. Denote the length of the longest minimal improvement path from any outcome to a Nash equilibrium L_Γ. The rest of this proof involves showing that as $t \to \infty$, the probability that the complete longest minimal improvement path has been traversed goes to 1.

In a game consisting of N agents using DSA–B to adapt their state, for any probability of updating $p \in (0,1)$, the probability that only one agent changes state at a particular time step is given by: $p(1-p)^{N-1}$. Consider the probability that at some time step, the agent selected to change its state is able to improve its utility (i.e. is part of an improvement path). This probability is at least $p(1-p)^{N-1}/N$, which is its value when at that time step, the improvement step is unique. Thus, at any time, t, the probability of traversing the longest minimal improvement path of length L_Γ, is at least:

$$q = \begin{cases} 0 & t < L_\Gamma, \\ \left[\frac{p(1-p)^{N-1}}{N}\right]_\Gamma^L & t \geq L_\Gamma. \end{cases}$$

Note that whenever $t \geq L_\Gamma$, q is greater than zero, because $p \in (0,1)$ and N and L_Γ are finite. Following this, in a sequence of t steps, the probability of traversing the longest minimal improvement path and converging to a Nash equilibrium at time t, $p_{conv}(t)$, simply follows a geometric distribution, with positive probabilities beginning at time step L_Γ:

$$p_{conv}(t) = q(1-q)^t$$

Consequently, we can express the cumulative probability of converging by t, $P_{conv}(t)$ — the sum of $p_{conv}(t)$ — as:

$$\begin{aligned} P_{conv}(t) = \sum_{\tau=1}^{t} p_{conv}(\tau) &= \frac{q(1-(1-q)^{t+1})}{1-(1-q)} \\ &= 1-(1-q)^{t+1} \end{aligned}$$

Then, as $t \to \infty$, $(1-q)^{t+1} \to 0$, so the probability that a complete longest minimal improvement path is traversed goes to 1 as the number of rounds tends towards infinity. □

The convergence proof for DSA–A follows the same argument, except that all Nash equilibria are absorbing. For BR–I, the proof is identical. This is because the only difference between DSA–B and BR–I is that the former selects the best response while the latter selects a better–response, and these cases are treated in the same way with respect to the finite improvement property: That is, the finite improvement property ensures that all improvement paths are finite, whether they be best–response or better–response paths.

We have now described two methods of proving the convergence of a DCOP algorithm. The first shows that an algorithm is anytime, and that it improves until it reaches a Nash equilibrium. The second technique begins by characterising the absorbing states of an algorithm. Then, by the finite improvement property, at any time in the future there is some non–negative probability of the algorithm entering the absorbing state. Therefore, the algorithm almost surely converges.

The algorithms discussed so far have produced individual best or better responses. However, one common drawback of these approaches is that if these algorithms converge to sub-optimal equilibria, they can not escape. One technique used to get around this problem is to use stochasticity in the decision rule. A second is to aggregate agents' utilities and allow coordinated, joint changes in state. These two techniques are discussed in the following two sections.

4.1.3 Convergence of SAP and DSAN to the Global Optimum

Both SAP and DSAN use a stochastic decision rule to escape from local maxima. Specifically, by using the logistic or simulated annealing decision rules, they can move between basins of attraction of local maxima.

In potential games, SAP is known to converge to a distribution that maximises the function:

$$\sum_{s \in S} u_g(s) Pr(s) - \eta \sum_{s \in S} Pr(s) \log Pr(s)$$

which is given by the Boltzmann distribution with temperature parameter η (Young, 1998, Chapter 6). By setting η low, this algorithm approximates the optimal joint state, and at any point in time has a high probability of having the optimal configuration.

Furthermore, regarding both SAP and DSAN, when the temperature parameter of these decision rules are decreased over time according to an appropriate annealing schedule (i.e., $\eta \propto 1/\log t$), they are known to converge to the Nash equilibrium that maximises the potential function (Kirkpatrick et al., 1983; Young, 1998; Benaïm et al., 2005, 2006). That is, they converge to the global optimum.

*4.1.4 Convergence of MGM–*k *and SCA–*k *to* k*–optima*

A second technique used to escape local maxima is to use a target function that aggregates local utilities to evaluate joint strategy changes by teams of agents. This is the approach used by MGM–2 and SCA–2, which both check for all joint changes in state by pairs of agents (as in Equation 11), and the families of MGM–*k* and SCA–*k* algorithms generally.

Similar to MGM, under MGM–2, only isolated pairs of agents act at a given step, so any change only improves the global utility, and the algorithm only terminates when it reaches a 2–optimum, rather than a Nash equilibrium (Maheswaran et al., 2005). The almost–sure convergence of SCA–2 is proven using the same method as DSA, except that the absorbing states are the set of 2–optima. However, note that bounds on worst–case 2–optima only exist for DCOPs containing unary and binary constraints, so the benefits of using MGM–2 in DCOPs with constraint arity greater than 2 are unclear. Nonetheless, these proofs can easily be extended to convergence to *k*–optima for the corresponding algorithms.

4.2 Algorithms Using Averaging Target Functions

In this section, we discuss averaging algorithms that use variations of the expected payoff over historical frequencies of actions and average regret target functions. We begin with the fictitious play family of algorithms, before considering regret–based algorithms.

4.2.1 Fictitious Play Algorithms

The term 'fictitious play' is often used to denote a family of adaptive processes that use the expected payoff over historical frequencies of actions as a target function (Fudenberg and Levine, 1998). 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.

The standard fictitious play algorithm, described in Table 1 as FP, uses the expected payoff over historical frequencies as a target function (Equation 6) and the *argmax*–B decision rule, and agents follow the flood schedule and adjust their state simultaneously. A proof of the convergence of FP to Nash equilibrium in weighted and exact potential games is given by Monderer and Shapley (1996a). Specifically, in repeated potential games, this algorithm *converges in beliefs*; that is, each agents' estimate of its opponents' strategies, which are used to calculate each of its own strategies' expected payoffs, converge as time progresses. This process induces some stability in an agent's choice of strategy because an agent's current strategy is based on its opponents' average past strategies, which means that an agent's belief moves through its belief space with decreasing step size. Consequently thrashing and cycling behaviour is reduced, compared to, say, DSA or the best–response dynamics.

The same target function and adjustment schedule are used in smFP as in FP, but, typically, the multinomial logit decision rule substitutes for the *argmax* rule. However, unlike SAP or DSAN, this substitution does not imply that the algorithm converges to the global maximum of the potential function. Rather, smFP converges to a Nash equilibrium, in much the same way as FP (Hofbauer and Sandholm, 2002). Nonetheless, in practice, using the logit decision rule does, on average, produce better quality solutions than the *argmax* rule. Leslie and Collins (2006) show how to analyse smFP when the temperature parameter reduces over time and smFP approximates FPin the limit.

The dynamics of all versions of joint–strategy fictitious play (JSFP) are quite different to that of FP. Specifically, strict Nash equilibria are absorbing for any algorithm that uses the JSFP target function (Equation 7) and *argmax*–A 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.

To date, convergence to Nash equilibrium has not been shown for a version of JSFP that operates on the flood schedule. However, regarding the version that operates on the parallel random schedule, JSFP *with inertia* (JSFP–I), its proof of convergence to strict Nash equilibria is based on a similar argument to that for the convergence of DSA (Marden et al., 2009b). 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, as with DSA, when inertia is added to the agents' choice of action (i.e. 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. As with FP, because agents' current strategies are based on average past joint strategies, the JSFP–I process produces relative stability in an agent's choice of strategy, and as a consequence thrashing and cycling behaviour is reduced. Additionally, because JSFP–I uses the parallel random schedule, the number of messages required each time step to run the algorithm is less than FP.

4.2.2 Adaptive Play Algorithms

The AP variants we consider here are all of those in which an agent takes a sample of size k from a finite memory of the previous m plays of the game to evaluate their expected rewards for state (Equation 8) and chooses a state using the ε–greedy choice rule, and all of the agents operate using the flood schedule (note that this excludes SAP). A subset of these algorithms can be shown to converge to a strict Nash equilibrium, using results from perturbed Markov processes (Young, 1993, 1998). The key elements of the proof are as follows.

First, call the particular joint memory maintained by the agents at t, the *memory configuration*. Note that if $\varepsilon = 0$, then the memory configurations containing only strict Nash equilibria are absorbing for any $k \leq m$. That is, with no random play, if all agents' memories contain only a single strict Nash equilibrium, that equilibrium will be played from there on. Second, using a *resistance tree* argument (Young, 1993), it can be shown that from any memory configuration, for any $1/|\nu(i)|\varepsilon > 0$ (where $\nu(i)$ are i's neighbours), the probability of moving along an improvement path towards a strict Nash equilibrium is greater than that for a movement away. As such, over time the probability of traversing an entire (finite) improvement

path goes to 1. This result holds provided that the sample size $k \leq {}^{m}/_{L_\Gamma+2}$, where L_Γ is the longest minimal improvement path from any joint–action profile to a strict Nash equilibrium. Building on this, as $\varepsilon \to 0$, the probability of the memory configuration consisting entirely of one strict Nash equilibrium also goes to 1. Then, in the limit, this strict Nash equilibrium is absorbing.

4.2.3 Regret Matching and Weighted Regret Monitoring Algorithms

Like the variations of fictitious play, algorithms that use the average regret for past actions to evaluate states also come in many different forms. Here we limit our attention to the regret matching (RM) and weighted regret monitoring with inertia (WRM–I) algorithms, which show contrasting behaviour as a result of a small difference in the target function they employ.

RM uses the average regret for past actions (Equation 9) in conjunction with a linear probabilistic decision rule (which assigns zero probability to strategies with negative regret values) to decide on a strategy, with agents adjusting by the parallel random schedule. RM converges to the set of correlated equilibria (a generalisation of Nash equilibria) in all finite games (Hart and Mas-Colell, 2000), however, it does not necessarily converge to Nash equilibria. Nonetheless, it is easy to see that by using this target function, an agent's worst performing strategies are ruled out earliest, and although the use of a linear probabilistic decision rule does cause some thrashing, the presence of negative regrets lessens these effects.

On the other hand, WRM–I does converge to a pure–strategy Nash equilibrium in potential games (Arslan et al., 2007). This algorithm uses a target function that discounts past regrets by a constant weight (Equation 10) and 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$^+$). In the case of the linear$^+$ decision rule, WRM–I differs from RM only in the target function used. 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 (Equation 10). 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. Additionally, 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 in the proof of 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 (as is the case in RM).

5 Conclusions

In this paper, we focused on local approximate best–response algorithms for DCOPs, for optimisation in domains where communication is difficult, costly or impossible, and in which optimality can be traded off against timeliness or computational and communicational burden. Specifically, our key contribution is a framework for analysing local approximate best–response algorithms for DCOPs — that is, algorithms that operate by having agents exchange messages that contain only their strategy. Our framework captures many algorithms developed in both the computer science and game theory literatures. Moreover, we argue that the appropriate solution concept for the class of local approximate best–response algorithms is the Nash equilibrium condition. Given this, our framework is built on the insight that when formulated as noncooperative games, DCOPs form a subset of the class of potential games. In turn, this allowed us to apply game theoretic methods to analyse the convergence properties of local approximate best–response algorithms developed in the computer science literature.

In general, our framework is based on a three stage decomposition that is common to all local approximate best–response DCOP algorithms. Given an appropriate trigger, an individual agent enters a state evaluation stage, which produces some measure of the desirability of each state. This is followed by a decision on which action to take, based on the preceding state evaluations. Then, the system–wide process that controls which agent adjusts its state at each point is given by an adjustment schedule. We populate our framework with algorithm components, corresponding to the three stages above, that are used in existing algorithms, and which can be used to construct novel algorithms.

Our framework can assist system designers by making the pros and cons of the various DCOP algorithm configurations clear. To illustrate this, we constructed three novel hybrid algorithms from the components identified in our parameterisation. We evaluated these hybrids alongside eight existing algorithms taken from both the computer science and game theory literatures. Our experimental results show that an algorithm's behaviour is accurately predicted by identifying its constituent components. For example, algorithms that use fictitious play–like target functions and an *argmax* decision rule converge to a Nash equilibrium, but by varying the adjustment schedule, a designer may trade off between convergence time and communication use. Thus, a system designer may use our framework to tailor a DCOP algorithm to suit their mix of requirements, whether they be high quality solutions (but, for example, in the presence of bandwidth restrictions), rapid convergence (such as in real–time settings), or low communication costs (e.g. in the presence of resource constraints such as battery life). Furthermore, we expect most of our experimental results to generalise to other problems that fall within the class of hypergraphical potential games.

Generally in field of DCOPs, the main problems requiring attention involve extending the basic, static model with known payoffs and lossless communication to encompass the real–world aspects of typical DCOP application domains. In more detail, the most salient of these aspects can be broken into the following groupings:

Online learning of unknown rewards: Online learning of reward functions poses a difficult problem in DCOPs, particularly if coordinated search of the joint–action space is not possible. It is important to consider the differences in approaches to the problem that are needed if the goal is to maximise the long–term reward (as is often addressed in Markov decision processes) or to find a 'good enough' solution quickly (as in optimal stopping problems). This problem can be further extended by considering the case were rewards are not just unknown, but observations of them are noisy, or even stochastic.

Dynamic problems: DCOPs have proven to be very useful for describing static problems, but their usefulness for dynamic and stochastic problems is not clear. There is, however, scope for exporting techniques for DCOPs to decentralised Markov decision processes and partially–observable Markov decision processes in order to identify tractable classes of those problems and to, subsequently, develop algorithms based on DCOP solution techniques. Furthermore, if decentralised optimisation mechanisms that produce timely solutions are desirable in many static scenarios, then there is an even greater demand for principled decentralised approximation heuristics for real–time sequential decision–making in dynamic scenarios, and we believe the approaches developed here represent a first step in developing such techniques.

Communication: The model of communication adopted in this paper is a natural, although naive one. Communication in real–world applications of DCOPs is lossy, noisy, delayed and otherwise asynchronous, and has not been systematically addressed. Similarly, we assume communication takes place over a network defined by the constraint graph. How relaxing this assumption, to consider cases where agents do not have a direct communication link with all of the agents their utility depends on, affects the efficacy of existing approaches is unknown.

Appendix A.

The following pseudocode describes several of the algorithms discussed in this paper. 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 sub–script *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 $u(k, s_\nu)$. The algorithms listed here are the maximum-gain messaging algorithm (MGM), the distributed stochastic algorithm using the *argmax*–B decision rule (DSA–B), 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$, s_\nu)$
for k = 1:K
  stateGain(k) = $u(s$=$k, s_\nu) -$ currentReward
end for
bestGainState = $\underset{k}{\text{argmax}}$ [stateGain]
bestGainValue = stateGain(bestStateGain)
sendBestGainMessage[allNeighbours, bestGainValue]
neighbourGainValues = getNeighbourGainValues[allNeighbours]
if bestGainValue $>$ max[neighbourGain] **then**
  newState = bestGainState
  sendStateMessage[allNeighbours, newState]
end if

DISTRIBUTED STOCHASTIC ALGORITHM (DSA–B)

currentValue = $u(s$= currentState$, s_\nu)$
for k = 1:K
  stateRegret(k) = $u(s$=$k, s_\nu) -$ currentValue
end for
candidateState = $\underset{k}{\text{argmax}}$ [stateRegret]
if rand[0,1] $\leq p$
  newState = candidateState
end if
if newState $\neq$ currentState
  sendStateMessage[allNeighbours, newState]
end if

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 state in a random sequence. In practice, there are a number of ways to implement this type of schedule, however, the simplest is to randomly select an agent to run the stated procedure. Note this usually means some agents may be given more than one opportunity to adjust their state in a particular time step, while other agents may have none.

SPATIAL ADAPTIVE PLAY (SAP)

currentValue = $u(s{=}\text{ currentState}, s_{\text{v}})$
for k = 1:K
  stateRegret(k) = $u(s{=}k, s_{\text{v}}) -$ currentValue
end for
for k = 1:K
  statePropensity(k) = $\exp[\eta^{-1}\text{stateRegret(k)}]$
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, $|\text{v}|$ 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 state 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 neighours' joint–strategy space.

FICTITIOUS PLAY (FP)

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

SMOOTH FICTITIOUS PLAY (SMFP)

for j = 1:|v|
  $\mathbf{q}_j^t = \frac{1}{t}[I\{k_j = s^t\} + (t-1)\mathbf{q}_j^{t-1}]$
end for
$t = t+1$
for k = 1:K
  for h = 1:H
    $\mathbb{E}[u(s=k, s_{\mathrm{v}}^h)] = u(s, s_{\mathrm{v}}^h) \prod_{s_j^h \in s_{\mathrm{v}}^h} q_j^t$
  end for
  stateValue(k) = $\sum_{h=1}^{H} \mathbb{E}[u(s=k, s_{\mathrm{v}}^h)]$
end for
for k = 1:K
  statePropensity(k) = exp[η^{-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) = $\frac{1}{t}[u(s{=}k, s_{\mathrm{v}}) + (t-1)$stateValue(k)]
end for
$t = t+1$
candidateState = $\underset{k}{\text{argmax}}$ [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$, s_{\mathrm{v}})$
for k = 1:K
  avgDiff(k) = $\rho u(s{=}k, s_{\mathrm{v}} -$ currentValue $+(1-\rho)$avgDiff(k))
  stateRegret(k) = max[avgDiff(k)$,0$]
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

References

S. M. Aji and R. J. McEliece. The generalized distributive law. *IEEE Transactions on Information Theory*, 46:325–343, 2000.

S. P. Anderson, A. de Palma, and J. Thisse. *Discrete Choice Theory of Product Differentiation*. MIT Press, Cambridge, MA, USA, 1992.

K. R. Apt. *Principles of Constraint Programming*. Cambridge University Press, Cambridge, UK, 2003.

M. Arshad and M. C. Silaghi. Distributed simulated annealing and comparison to DSA. In *Proceedings of the 4th International Workshop on Distributed Constraint Reasoning (DCR–03)*, Acapulco, Mexico, 2003.

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

R. J. Aumann. Acceptable points in general cooperative n-person games. In A. W. Tucker and R. D. Luce, editors, *Contributions to the Theory of Games IV*, pages 287–324. Princeton University Press, Princeton, NJ, USA, 1959.

M. Benaïm, J. Hofbauer, and S. Sorin. Stochastic approximation and differential inclusions. *SIAM Journal of Control and Optimisation*, 44(1):328–348, 2005.

M. Benaïm, J. Hofbauer, and S. Sorin. Stochastic approximations and differential inclusions, part II: Applications. *Mathematics of Operations Research*, 31(4):673–695, 2006.

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

A. Blum and Y. Mansour. From external to internal regret. *Journal of Machine Learning Research*, 8: 1307–1324, June 2007.

E. Bowring, J. Pearce, C. Portway, M. Jain, and M. Tambe. On *k*–optimal distributed constraint optimization algorithms: New bounds and algorithms. In *Proceedings of the 7th International Conference on Autonomous Agents and Multiagent Systems (AAMAS–08)*, pages 607–614, Estoril, Portugal, 2008.

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

A. Chapman, R. A. Micillo, R. Kota, and N. Jennings. Decentralised dynamic task allocation: A practical game–theoretic approach. In *The 8th International Conference on Autonomous Agents and Multiagent Systems (AAMAS–09)*, pages 915–922, 2009.

M. Cooper, S. de Givry, and T. Schiex. Optimal soft arc consistency. In *Proceedings of the 20th Internation Joint Conference on Artificial Intelligence (IJCAI–07)*, pages 68–73, 2007.

V. P. Crawford. Adaptive dynamics in coordination games. *Econometrica*, 63:103–143, 1995.

A. Farinelli, A. Rogers, A. Petcu, and N. R. Jennings. 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)*, pages 639–646, 2008.

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

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

D. Fudenberg and D. K. Levine. Consistency and cautious fictitious play. *Journal of Economic Dynamics and Control*, 19:1065–1089, 1995.

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

B. P. Gerkey and M. J. Mataric. Sold!: Auction methods for multirobot coordination. *IEEE Transactions on Robotics and Automation*, 18(5):758–768, Oct 2002.

G. Grimmett and D. Stirzaker. *Probability and Random Processes*. Oxford University Press, third edition, 2001.

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

S. Hart and A. Mas-Colell. A reinforcement procedure leading to correlated equilibrium. In G. Debreu, W. Neuefeind, and W. Trockel, editors, *Economic Essays: A Festschrift for Werner Hildenbrand*, pages 181–200. Springer, New York, NY, USA, 2001.

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

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

K. Hirayama and M. Yokoo. The distributed breakout algorithms. *Artificial Intelligence*, 161(1-2):89–115, 2005.

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

J. Kho, A. Rogers, and N. Jennings. Decentralised control of adaptive sampling in wireless sensor networks. *ACM Transactions on Sensor Networks*, 5(3), 2009.

S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimisation by simulated annealing. *Science*, 220: 671–680, 1983.

H. Kitano, S. Todokoro, I. Noda, H. Matsubara, T. Takahashi, A. Shinjou, and S. Shimada. 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)*, volume 6, pages 739–743, Tokyo, Japan, 1999.

M. Krainin, B. An, and V. Lesser. 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)*, pages 138–145, Fremont, CA, USA, November 2007. IEEE Computer Society Press.

F. R. Kschischang, B. J. Frey, and H. Loeliger. Factor graphs and the sum-product algorithm. *IEEE Transactions on Information Theory*, 47:498–519, 2001.

P. La Mura and M. R. Pearson. Simulated annealing of game equilibria: A simple adaptive procedure leading to Nash equilibrium. In *International Workshop on The Logic and Strategy of Distributed Agents*, Trento, Italy, 2002.

D. S. Leslie and E. J. Collins. Generalised weakened fictitious play. *Games and Economic Behavior*, 56: 285–298, 2006.

R. T. Maheswaran, J. P. Pearce, and M. Tambe. Distributed algorithms for DCOP: A graphical-game-based approach. In *Proceedings of the 17th International Conference on Parallel and Distributed Computing Systems (PDCS–04)*, pages 432–439, San Francisco, CA, USA, 2004.

R. T. Maheswaran, J. P. Pearce, and M. Tambe. A family of graphical-game-based algorithms for distributed constraint optimization problems. In *Coordination of Large-Scale Multiagent Systems*, pages 127–146. Springer-Verlag, Heidelberg, Germany, 2005.

R. Mailler and V. Lesser. Asynchronous partial overlay: A new algorithm for solving distributed constraint satisfaction problems. *Journal of Artificial Intelligence Research*, 25:529–576, 2006.

J. R. Marden, G. Arslan, and J. S. Shamma. Connections between cooperative control and potential games illustrated on the consensus problem. *IEEE Transactions on Systems, Man and Cybernetics, Part B: Cybernetics*, in press, 2009a.

Jason R. Marden, Gürdal Arslan, and Jeff S. Shamma. Joint strategy fictitious play with inertia for potential games. *IEEE Transaction on Automatic Control*, in press, 2009b.

G. M. Matthews and H. F. Durrant-Whyte. Scalable decentralised control for multi-platform reconnaissance and information gathering tasks. In *Proceedings of the 9th International Conference on Information Fusion (Fusion'06)*, 2006.

N. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller, and E. Teller. Equation of state calculations by fast computing machines. *Journal of Chemical Physics*, 21:1087–1092, 1953.

C. Mezzetti and J. W. Friedman. Learning in games by random sampling. *Journal of Economic Theory*, 98(1):55–84, 2001.

P. Jay Modi, W. Shen, M. Tambe, and M. Yokoo. ADOPT: Asynchronous distributed constraint optimization with quality guarantees. *Artificial Intelligence*, 161(1-2):149–180, 2005.

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

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

P. Morris. The breakout method for escaping from local minima. In *Proceedings of the 11th National Conference on Artificial Intelligence (AAAI '93)*, pages 40–45, Washington, DC, USA, 1993.

C. H. Papadimitriou and T. Roughgarden. Computing correlated equilibria in multi–player games. *J. ACM*, 55(3):14, 2008.

J. P. Pearce and M. Tambe. Quality guarantees on *k*-optimal solutions for distributed constraint optimisation problems. In *Proceedings of the 20th Internation Joint Conference on Artificial Intelligence (IJCAI–07)*, pages 1446–1451, Hyderabad, India, 2007.

A. Petcu and B. Faltings. DPOP: A scalable method for multiagent constraint optimization. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI–05)*, pages 266–271, Edinburgh, Scotland, Aug 2005.

W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling. *Numerical Recipes: The Art of Scientific Computing*. Cambridge University Press, 1992.

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

A. Rogers, D. D. Corkill, and N. R. Jennings. Agent technologies for sensor networks. *IEEE Intelligent Systems*, 24(2):13–17, March 2009.

T. Roughgarden. *Selfish Routing and the Price of Anarchy*. MIT Press, 2005.

T. Schiex, H. Fargier, and G. Verfaillie. Valued constraint satisfaction problems: Hard and easy problems. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI–95)*, pages 631–639, 1995.

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

R. Stranders, A. Farinelli, A. Rogers, and N. R. Jennings. Decentralised coordination of mobile sensors using the max-sum algorithm. In *Proceedings of the 21st International Joint Conference on Artifical Intelligence (IJCAI–09)*, 2009.

A. Stranjak, P. S. Dutta, M. Ebden, A. Rogers, and P. Vytelingum. 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)*, May 2008.

G. Tel. *Introduction to Distributed Algorithms*. Cambridge University Press, Cambridge, UK, 2000.

K. Tumer and D. H. Wolpert, editors. *Collectives and the Design of Complex Systems*. Springer, New York, NY, USA, 2004.

P. van Leeuwen, H. Hesselink, and J. Rohling. Scheduling aircraft using constraint satisfaction. In *Electronic Notes in Theoretical Computer Science*, pages 252–268. Elsevier, 2002.

Y. Weiss. Correctness of local probability propagation in graphical models with loops. *Neural Computation*, 12(1):1–41, 2000.

Y. Xu, P. Scerri, B. Yu, S. Okamoto, M. Lewis, and K. Sycara. An integrated token–based algorithm for scalable coordination. In *Proceedings of the 4th International Conference on Autonomous Agents and Multiagent Systems (AAMAS–05)*, pages 407–414, 2005.

M. Yokoo and K. Hirayama. Distributed breakout algorithm for solving algorithm for solving distributed constraint satisfaction and optimization problems. In *Proceedings of the 2nd International Conference on Multiagent Systems (ICMAS '96)*, pages 401–408, 1996.

H. P. Young. *Individual Strategy and Social Structure: An Evolutionary Theory of Institutions*. Princeton University Press, Princeton, NJ, USA, 1998.

H. Peyton Young. The evolution of conventions. *Econometrica*, 61:57–84, 1993.

W. Zhang and Z. Xing. Distributed breakout vs. distributed stochastic: A comparative evaluation on scan scheduling. In *Proceedings of the AAMAS–02 workshop on Distributed Constraint Reasoning*, pages 192–201, 2002.

W. Zhang, G. Wang, Z. Xing, and L. Wittenburg. Distributed stochastic search and distributed breakout: Properties, comparison and applications to constraint optimization problems in sensor networks. *Artificial Intelligence*, 161:55–87, 2005.