

A Parameterisation of Algorithms for Distributed Constraint Optimisation via Potential Games^{*}

Archie C. Chapman^{**}, Alex Rogers, and Nicholas R. Jennings

Electronics and Computer Science, University of Southampton, Southampton SO17 1BJ, UK.
(ac05r,acr,nrj)@ecs.soton.ac.uk

Abstract. This paper introduces a parameterisation of learning algorithms for distributed constraint optimisation problems (DCOPs). This parameterisation encompasses many algorithms developed in both the computer science and game theory literatures. It 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 algorithms developed in the computer science literature using game theoretic methods. Furthermore, our parameterisation can assist system designers by making the pros and cons of, and the synergies between, the various DCOP algorithm components clear.

1 Introduction

In this paper, we focus on algorithms for distributed constraint optimisation problems (DCOPs) for use in situations where the actors are distributed and can only communicate with their peers. Thus, we exclude centralised approaches in which all of the information needed to solve a problem is accessible to a single decision maker, and also exclude distributed algorithms that rely on highly structured interactions, such as algorithms that run on a pre-computed tree, because such interactions often become prohibitively costly as the size of the problem increases. We call the remaining algorithms *completely distributed* and, broadly speaking, these algorithms can be grouped into two sets based on their origins: (i) distributed heuristic processes taken from the game theory literature, and (ii) distributed versions of the centralised procedures developed for traditional constraint optimisation problems from the computer science literature.

Given this context, we show that DCOPs, when viewed from a game theory perspective, form a subset of the class of potential games [1], a useful class of games with several desirable properties. We use this insight to bring together the two sets of algorithms and analyse them under a single framework. In more detail, this paper advances the state of the art in the following ways. First, we show that DCOP games are a subset of potential games — a new result which allows us to apply established methods for analysing algorithms from game theory to existing algorithms produced by the computer science community. Second, we develop a parameterisation that encompasses the major completely distributed DCOP algorithms. This framework allows us to elucidate

^{*} 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.

^{**} Thanks to David Leslie for the many interesting discussions related to this work.

the relationships between the various algorithms in the form of a parameterisation of the DCOP algorithm design space. Third, by constructing such a unified view, we are able to uncover synergies that arise as a result of combining various approaches. We discuss the connections between, and overlapping features of, game-theoretic algorithms and algorithms developed by computer scientists specifically for solving DCOPs. In the process, we give two examples of how our parameterisation, guided by the properties of potential games, can be used to construct novel algorithms in a principled manner.

The paper progresses as follows. In the next section we give an overview of potential games, introduce DCOPs, and show that DCOP games are a subset of the class of potential games. Section 3 contains the main contribution of the paper: a parameterisation of completely distributed algorithms for DCOPs. In Section 4, we show how the major DCOP algorithms fit our parameterisation, and discuss the connections between these algorithms. The final section concludes and discusses future work.

2 DCOPs as Potential Games

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

An agent's goal is to maximise its own payoff, conditional on the choices of its opponents. Stable points in such a system are characterised by the set of Nash equilibria.

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

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

In a Nash equilibrium, no individual agent has an incentive to change their strategy. This concept is important because, in general, a Nash equilibrium in mixed strategies (probability distributions over pure strategies) is guaranteed to exist in all finite games. We can also define a *strict* Nash equilibrium, which is a necessary component of many convergence proofs in game theory, by replacing the inequality in Equation 1 with a strict inequality. The implication is that in a strict Nash equilibrium, no agent is indifferent between their equilibrium strategy and another strategy.

The class of potential games is characterised as those games that admit a potential function on the joint strategy space whose gradient is the gradient of the constituents' private utility functions [1]. A potential function has a natural interpretation as representing opportunities for improvement to a player defecting from any given strategy profile. Thus, the local optima of the potential function are Nash equilibria of the game.

The class of finite potential games are used to describe many problems in multi-agent systems, in particular congestion problems on networks [2], and more recently, power control problems, channel selection problems and scheduling problems in wireless networks [3, 4], as well as target assignment problems [5] and job scheduling [6]. We now formally define a potential game.

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.

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

First, every finite potential game possesses at least one pure-strategy equilibrium [2]. To see this, let P be a potential for a game. Then s is an equilibrium point for the potential game if and only if for every $i \in N$,

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

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

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

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

Then, as S is a finite set, the sequence ϕ must be finite, so every potential has the finite improvement property. The finite improvement property ensures that the behaviour of agents who independently play 'better-responses' in each period of the repeated game converges to a Nash equilibrium. Taken together, the two properties discussed above ensure that many simple adaptive processes converge to a pure-strategy Nash equilibrium in a repeated potential game (discussed in Section 4).

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

A constraint optimisation problem is formally represented by a set of variables $X = \{x_1, x_m, \dots\}$, each of which may take one of a finite number of states or values, $s_j \in S_j$, a set of constraints $C = \{c_1, c_2, \dots\}$, and a global utility function, u_g , that specifies preferences over configurations of states of variables in the system. A constraint $c = \langle X_c, R_c \rangle$ is defined over a set of variables $X_c \subset X$ and a relation between those variables, R_c , which is a subset of the Cartesian product of the domains of each variable involved in the constraint, $\prod_{x_j \in X_c} S_j$. A function that specifies the reward for satisfying,

or penalty for violating, a constraint is written $u_{c_k}(s_{c_k})$, where s_{c_k} is the configuration of states of the variables X_{c_k} . Using this, the global utility function aggregating the utilities from satisfying or violating constraints commonly takes the form:

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

Importantly, this aggregation is strictly monotonic, in that an increase in the number of satisfied constraints results in an increase in the global utility. Constraints may be ascribed different levels of importance by simply weighting the rewards for satisfying them, or by using a positive monotonic transform of constraint reward [7]. The objective is then to find a global configuration of variable states, s^* , such that:

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

Given this, a DCOP is produced when a set of autonomous agents each control the state of a subset of the variables. A DCOP game is a simple formulation that explicitly models the strategic dependencies between the variables each agent controls [8]. Without loss of generality, we consider the case where each agent controls only one variable, so we use the terms ‘state of a variable’ and ‘strategy of an agent’ interchangeably. We notate the set of agents involved in a constraint by N_c , the set of constraints that i is involved by C_i , and the agents that i shares constraints with, i ’s *neighbours*, by $v(i)$.

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

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

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

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

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

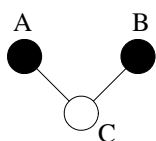
Proof. A change in i ’s strategy only affects i ’s neighbours, $v(i)$, so the following hold:

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

where the third line flows from the second by definition. \square

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

In the case of binary constraints, the game played between pairs of agents sharing a constraint can be easily expressed in matrix form. One widely studied binary constraint optimisation problem is graph colouring. In graph colouring, neighbouring nodes share constraints, which are satisfied if the nodes are in differing states. Consider the following graph colouring problem, where each node can be either black or white, and the associated constraint game:



$$u_c(s_i, s_j) = \begin{array}{c|cc} & \text{B} & \text{W} \\ \hline \text{B} & (0, 0) & (1, 1) \\ \hline \text{W} & (1, 1) & (0, 0) \\ \hline \end{array}$$

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

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

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

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

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

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

3 The DCOP Algorithm Design Space

In this section we describe the basic components of many DCOP algorithms present in the literature. By doing this, we open a way to investigate synergies that arise by com-

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

Given this background, this section examines the forms that each of the three algorithm stages can take. 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 (BR) and smooth best response (smBR) [9]; the distributed stochastic algorithm (DSA) [10]; distributed simulated annealing (DSAN) [11]; the maximum-gain messaging algorithm (MGM) [12, 8]; fictitious play (FP) [13] and smooth fictitious play (smFP) [14, 9]; regret matching (RM) [15]; spatial adaptive play (SAP) [16, Chapter 6]; and, the stochastic coordination-2 (SCA-2) and maximum-gain messaging-2 algorithms (MGM-2) [8].

We also note that three significant DCOP algorithms — asynchronous partial overlay (OptAPO) [17], asynchronous decentralised optimisation (ADOPT) [18] and dynamic programming optimisation (DPOP) [19] — are not included in this parametrisation.¹ Nonetheless, we do refer to techniques used by these algorithms that may be applied to distributed learning algorithms.

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

3.1 State Evaluations

The way in which a DCOP 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 next subsection addresses using immediate payoffs as a target function, while subsequent ones examine the more sophisticated alternatives.

Immediate payoff. 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

¹ As noted in the introduction, OptAPO is excluded as it proceeds by centralising the DCOP problem, and so is not comparable to the algorithms included. ADOPT and DPOP are excluded as they do not use adaptive or learning heuristics, and they rely on a significant preprocessing stage that involves the construction of a depth-first search tree on which the algorithm is run.

typical ‘hill-climbing’ or ‘greedy’ behaviour. This leads the system to a Nash equilibrium, which corresponds to a local potential–maximising point. Furthermore, many algorithms, including DSA, DSAN and MGM, use the amount to be gained by changing state as a target function. This is a simple perturbation of the utility function, and for many decision rules, using either the gain or the raw utility function as an input will produce the same result.

Agents using this target function to update their state evaluations 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.

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. In this case, let i ’s belief over its opponents’ joint strategy profiles, $q_i^t(s_{-i})$, be given by the frequency with which it observes each joint profile. Each agent’s expected payoff given this belief, written FP_i^t , is then:

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

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

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

where $u_i(s_i, s_{-i}^t)$ is i ’s payoff for each element of S_i given its opponents’ profile at t .

FP and smFP use this measure of expected payoff as a target function. It has the same communication requirements as algorithms that use the immediate payoff as a target function, because at each point in time each agent only needs to know the values of $u_i(s_i, s_{-i})$ for each of its states. Variations of fictitious play that use other methods to update the agent’s belief state have been suggested, such as *cautious* fictitious play, for situations where an agent can only observe its payoff and so uses the average received payoff to each action [20], or *weighted* fictitious play, for highly variable environments, in which past observations’ contribution to an agent’s belief are exponentially discounted [21].

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, written AR_i^t , where regret is the difference in payoff for choosing a state and the state that was actually chosen at a particular time:

$$AR_i^t = \frac{1}{t} \sum_{\tau=1}^t [u_i(s_i, s_{-i}^\tau) - u_i(s_i^\tau, s_{-i}^\tau)] \quad (2)$$

This target function is also known as *external regret*. Like the measure of expected payoff 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:

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

This target function is used by [15] to construct the RM algorithm, and is generalised to characterise an entire class of adaptive strategies [22]. It is also used as the target function for a distributed simulated annealing method for finding the Nash equilibria of games [23]. Average regret uses the same observations as algorithms that use the immediate payoff or expected payoff for an action as a target function. Like fictitious play, many variations of the method of updating regrets have been suggested. For example, a variation of the average regret measure, for situations where an agent can only observe its own payoff, is known as *internal regret* [24]. 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. Another example is a target function with exponentially discounted regrets [5].

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 probabilistic decision rule, as will be discussed in Section 3.2). A number of algorithms avoid this problem by using aggregated payoffs to evaluate states. These algorithms have significantly increased communication requirements, because agents pass information regarding the value of each state, rather than just indicating the current state of their variables.

MGM-2 and SCA-2 both use a pairwise aggregate of local utility functions to evaluate the joint state of two neighbouring agents, i and j :

$$u_{ij} = u_i(s_i, s_j, s_{-i,j}) + u_j(s_i, s_j, s_{-i,j}) \quad (3)$$

This allows the agents to synchronise their state changes, and can be used to avoid the worst Nash equilibria in the system by converging only to 2-optima. In general, a k -optimum is a strategy profile that is stable in the face of deviations of all coalitions of size k and less. In [25], the worst-case k -optimum solution to a DCOP game is shown to be better than the worst-case $(k-1)$ -optimum. This result implies that an algorithm that uses a pairwise aggregated target function has a higher lower-bound on its solution than any algorithm that only converges to a Nash equilibrium.

Furthermore, [8] 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. We now consider these rules in more detail.

Argmax and argmin decision rules. The *argmax* function (or, equivalently, the *argmin* function) returns the state with the highest (lowest) valued target function. If two or more states achieve the same maximum value, typically one is chosen with equal probability. However, in the case that one of those states is the current state, most implementations maintain that state and do not randomise. In this case, an algorithm using the immediate reward as a target function will sometimes converge to a Nash equilibrium that is not strict. In the other case, a similar algorithm would only ever converge to a strict Nash equilibrium.

A benefit of using the *argmax* function in conjunction with an immediate reward target function is that the resulting algorithm usually converges quickly, and depending on specifics, may even be anytime (e.g. MGM). However, one drawback of this technique is its dependence on initial conditions. It is quite 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. On the other hand, adding ergodicity to an algorithm by using a probabilistic decision rule allows it 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 and usually increasing the convergence time.

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:

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

This model is only appropriate when the target function supplies a non-negative input. For example, RM uses the linear probabilistic choice function with negative regrets set equal to zero, so that they are chosen with zero probability [15]. However, in general, this is quite a substantial limitation which limits the applicability of the linear probabilistic choice rule.

Multinomial logit decision rules. One probabilistic decision rule that can accept negative input is the multinomial logit decision rule or Boltzmann distribution [26]:

$$P_{s_i}(\eta) = \frac{e^{\eta^{-1} u_i(s_i, s_{-i}^t)}}{\sum_{s_i \in \mathcal{S}_i} e^{\eta^{-1} u_i(s_i, s_{-i}^t)}}.$$

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. The later case is referred to in the online reinforcement learning literature as a ‘greedy in the limit with infinite exploration’ decision rule [27]. This decision rule is used in typical specifications of smBR, SAP and smFP.

Simulated annealing decision rules. The simulated annealing decision rule is a probabilistic decision rule that works by randomly selecting a new candidate state and accepting or rejecting it based on a comparison to the current state [28, 29]. 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:

$$P_{s_i}(\eta) = \begin{cases} 1 & \text{if } u_i(s_i, s_{-i}) \geq u_i(j, s_{-i}) \\ e^{\eta^{-1}(u_i(j, s_{-i}) - u_i(s_i, s_{-i}))} & \text{else} \end{cases}$$

where $u_i(j, s_{-i})$ is the current state’s payoff. As with the multinomial logit choice model, η is a temperature parameter. If $\eta = 0$ then only states that improve the target function are accepted, while if $\eta = \infty$ all candidate states are accepted, and the state of the system follows a random walk. The temperature may be kept constant, resulting in an analogue of the Metropolis algorithm [28], or may be decreased over time as in a standard simulated annealing optimisation algorithm [29]. This rule is used in the DSAN algorithm, and a simulated annealing algorithm based on average regret has been suggested as a computational technique for solving for the Nash equilibria of general games [23].

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 their strategy at a given point in time.

Flood schedule. Under the flood schedule, all agents adjust their strategies at the same time. It is frequently used in applications of local greedy algorithms, and in DSAN, FP and smFP. A problem commonly observed with algorithms using the flood schedule, particularly greedy algorithms, is the presence of ‘thrashing’ or cycling behaviour [6]. 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 can be detrimental to the performance of any algorithm using the flood schedule.

Parallel random schedules. Parallel random adjustment schedules are simply variations of the flood schedule, in which each agent has some probability p of actually changing their state at any time step. As such, the parameter p is known as the degree of parallel executions [6]. These types of adjustment schedules do not ensure that no

thrashing takes place, but by selecting an appropriate value of p , or decreasing p along an appropriate schedule, thrashing and cycling behaviour can be minimised, producing an algorithm with parallel execution without increased communicational requirements. This is the adjustment schedule used by DSA and RM.

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 the finite improvement property of potential games, which directly implies that agents who play a sequence of ‘better responses’ converges to a Nash equilibrium in a finite number of steps. This property is used to prove the convergence of SAP and a sequential–move version of fictitious play [30].

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.

Maximum–gain priority adjustment schedule. The MGM algorithm takes its name from the type of adjustment schedule it uses [8, 12]. 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.

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 OptAPO algorithm.

4 DCOP Algorithm Parametrisation

In this section we show how many well known algorithms fit our parameterisation. Table 1 shows the parameterisation of seven main DCOP algorithms, which clearly shows the relationships between the algorithms, in terms of the components used to construct each algorithm.

Building in this, we give two examples of how the properties of potential games can be used to construct convergence proofs for several algorithms that share common components. First, we consider local best–response algorithms, which use the immediate reward as a target function and the *argmax* decision rule. In comparing two such algorithms, DSA and MGM, we highlight how small differences can affect an algorithm’s properties, and clearly demonstrate the benefits of a potential game characterisation of DCOPs. We also show how our parameterisation can be used to develop novel algorithms, by introducing a hybrid algorithm based on SAP. Second, we examine the fictitious play family of algorithms, which use the expected payoff over historical frequencies of actions as a target function, and show how our parameterisation allows for a straightforward generalisation of this family of algorithms.

	Target Function	Memory	Decision Rule	Adjustment Sched.
DSA	$u_i(s)$	—	argmax	Parallel random (p)
MGM	$u_i(s)$	—	argmax	Preferential: Maximum Gain
FP	$\frac{1}{t} \sum_{\tau=1}^t u_i(s_i, s_{-i}^\tau)$	Expected values	argmax	Flood
RM	$\frac{1}{t} \sum_{\tau=1}^t \begin{bmatrix} u_i(s_i, s_{-i}^\tau) \\ -u_i(s_i^\tau, s_{-i}^\tau) \end{bmatrix}$	Regret values	linear probabilistic	Parallel random (p)
SAP	$u_i(s)$	—	logistic	Sequential random
DSAN	$u_i(s)$	—	sim. annealing	Parallel random (p)
smBR	$u_i(s)$	—	logistic	Flood
smFP	$\frac{1}{t} \sum_{\tau=1}^t u_i(s_i, s_{-i}^\tau)$	Expected values	logistic	Flood
SCA-2	$u_i(s) + u_j(s)$	—	argmax	Parallel random (p)
MGM-2	$u_i(s) + u_j(s)$	—	argmax	Preferential: Maximum Gain

Table 1: Parameterisation of the main DCOP algorithms

4.1 Local Best-Response Algorithms

We begin by considering DSA and MGM, both of which use the immediate payoff for an action as their target function and the *argmax* function for their decision rule. They differ purely by the adjustment schedule they employ: DSA uses a random parallel schedule, and MGM a maximum-gain preference schedule. This difference affects the convergence properties of the algorithms. In more detail, MGM converges to a Nash equilibrium and is an anytime algorithm in potential games. This is because when agents act in isolation, their actions only ever improve their own utility, which implies an improvement in global utility (by Equation 2), and, by the same reasoning, the finite improvement property ensures that this algorithm converges to a Nash equilibrium in finite time.² However, DSA is not anytime, as it is possible for agents that change state at the same time to find themselves in a worse state than they began in, nor does it converge in finite time. Nonetheless, we can show the following, weaker,³ convergence result:

Theorem 2. *The distributed stochastic algorithm almost surely converges to a Nash equilibrium in repeated potential games.*

Proof. Sketch: A Nash equilibrium is an absorbing strategy profile under the DSA’s dynamics; that is, once in a Nash equilibrium, no agent will change their strategy. Now,

² In [8], MGM is shown to converge to an element in the set of Nash equilibrium and to be anytime in DCOP games directly, without using a potential game characterisation of the problem.

³ Almost sure convergence is a weaker form of convergence than finite time convergence, as at any point in time the system can be arbitrarily far from an equilibrium strategy profile.

for any non-Nash equilibrium outcome, there exists a longest improvement path, of length M , that terminates at a Nash equilibrium. In a game consisting of N agents using DSA 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: $p(1-p)^{N-1}$. Now, the probability that, at some time step, the selected agent is able to improve its utility is at least $p(1-p)^{N-1}/N$, which is its value when the improvement step is unique. Then, at any time, the probability of traversing the longest improvement path of length M , is at least:

$$\left[\frac{p(1-p)^{N-1}}{N} \right]^M$$

The proof is completed by using a geometric distribution to show that as $t \rightarrow \infty$, the probability that a complete worst-case improvement path is traversed goes to 1. \square

A similar convergence proof can be derived for a novel hybrid algorithm constructed by replacing the multinomial logit decision rule of SAP with the *argmax* function, which we call greedy spatial adaptive play. By the finite improvement property, greedy spatial adaptive play almost surely converges to a Nash equilibrium in potential games, because, as with DSA, the probability that a complete worst-case improvement path is completed goes to 1 as $t \rightarrow \infty$.

4.2 The Fictitious Play Family of Algorithms

The term ‘fictitious play’ is often used to denote a family of adaptive processes which use the expected payoff over historical frequencies of actions as a target function. This family significantly generalise the standard fictitious play algorithm [31]. The traditional version of fictitious play, described in Table 1 as FP, uses the flood schedule, so agents adjust their state simultaneously. Regarding FP, a proof of the convergence of this algorithm to Nash equilibrium in potential games is given by [32]. Additionally, a sequential-move version of fictitious play also converges to Nash equilibrium in non-degenerate potential games⁴ [30]. In more detail, in repeated potential games, these algorithms *converges in beliefs*; that is, each agent’s estimate of its opponents’ strategies, which is used to calculate each of its own strategy’s expected payoff, converges as time progresses.

Furthermore, any algorithm that uses historical frequencies as a target function and the *argmax* decision rule (regardless of the adjustment schedule used) has the property that if play converges to a pure-strategy profile, it must be a Nash equilibrium, because if it were not some agent would eventually change their strategy. Additionally, strict Nash equilibria are absorbing; if a strict Nash equilibrium is played once it is played from then on [9]. These results imply that the adjustment schedule is of little importance to the behaviour of algorithms in the fictitious play family, and so can be freely selected to best suit the situation at hand.

⁴ For our purposes, a non-degenerate game satisfies the condition that, for every $i \in N$ and for every $s_{-i} \in S_{-i}$:

$$u_i(s'_i, s_{-i}) \neq u_i(s''_i, s_{-i}).$$

That is, for any pure strategy profile of his opponents, an agent’s best response correspondence contains only one strategy.

5 Conclusions and Future Work

In this paper, we focus on completely distributed algorithms for DCOPs. Our key contribution is a parameterisation of completely distributed algorithms for DCOPs, which captures many algorithms developed in both the computer science and game theory literatures. This parameterisation is built on our insight that when formulated as non-cooperative games, DCOPs form a subset of the class of potential games. In turn, this allows us to apply game theoretic methods to analyse algorithms developed in the computer science literature. In Section 3 we showed how many existing algorithms fit our parameterisation. Then, in Section 4, we showed how they can be analysed using results concerning potential games, and also give two examples of how our parameterisation can be used to construct novel algorithms in a principled manner.

In [33], we extend this research by empirically comparing many of the algorithms, both hybrids and existing algorithms, considered in this paper. The experiments reported in that work examine the trade-offs between solution quality, convergence time and communication requirements for using different DCOP algorithm components.

References

1. Monderer, D., Shapley, L.S.: Potential games. *Games and Economic Behavior* **14** (1996) 124–143
2. Rosenthal, R.W.: A class of games possessing pure-strategy Nash equilibria. *International Journal of Game Theory* **2** (1973) 65–67
3. Hayajneh, M., Abdallah, C.T.: Distributed joint rate and power control game-theoretic algorithms for wireless data. *IEEE Communications Letters* **8** (2004) 511–513
4. Heikkinen, T.: A potential game approach to distributed power control and scheduling. *Computer Networks* **50** (2006) 2295–2311
5. Arslan, G., Marden, J.R., Shamma, J.S.: Autonomous vehicle-target assignment: A game theoretical formulation. *ASME Journal of Dynamic Systems, Measurement and Control* **129** (2007) 584–596
6. Zhang, W., Xing, Z.: Distributed breakout vs. distributed stochastic: A comparative evaluation on scan scheduling. In: *Proceedings of the AAMAS '02 workshop on Distributed Constraint Reasoning*. (2002) 192–201
7. Schiex, T., Fargier, H., Verfaillie, G.: Valued constraint satisfaction problems: Hard and easy problems. In: *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI '95)*. (1995) 631–639
8. Maheswaran, R.T., Pearce, J.P., Tambe, M.: A family of graphical-game-based algorithms for distributed constraint optimization problems. In: *Coordination of Large-Scale Multiagent Systems*. Springer-Verlag, Heidelberg, Germany (2005) 127–146
9. Fudenberg, D., Levine, D.K.: *The Theory of Learning in Games*. MIT Press, Cambridge, MA (1998)
10. Fitzpatrick, S., Meertens, L.: Distributed coordination through anarchic optimization. In Lesser, V., Jr., C.L.O., Tambe, M., eds.: *Distributed Sensor Networks: A Multiagent Perspective*. Kluwer Academic Publishers (2003) 257–295
11. Arshad, M., Silaghi, M.C.: Distributed simulated annealing and comparison to DSA. In: *Proceedings of the Fourth International Workshop on Distributed Constraint Reasoning (DCR '03)*, Acapulco, Mexico (2003)

12. Yokoo, M., Hirayama, K.: Distributed breakout algorithm for solving distributed constraint satisfaction and optimization problems. In: Proceedings of the Second International Conference on Multiagent Systems (ICMAS '96). (1996) 401–408
13. Brown, G.W.: Iterative solution of games by fictitious play. In Koopmans, T.C., ed.: *Activity Analysis of Production and Allocation*. Wiley, New York (1951) 374–376
14. Fudenberg, D., Kreps, D.: Learning mixed equilibria. *Games and Economic Behavior* **5** (1993) 320–367
15. Hart, S., Mas-Colell, A.: A simple adaptive procedure leading to correlated equilibrium. *Econometrica* **68** (2000) 1127–1150
16. Young, H.P.: *Individual Strategy and Social Structure: An Evolutionary Theory of Institutions*. Princeton University Press, New Jersey (1998)
17. Mailler, R., Lesser, V.: Solving distributed constraint optimization problems using cooperative mediation. In: Proceedings of the 3th International Conference on Autonomous Agents and Multiagent Systems (AMAAS '04). (2004) 438–445
18. Modi, P.J., Shen, W.M., Tambe, M., Yokoo, M.: ADOPT: Asynchronous distributed constraint optimization with quality guarantees. *Artificial Intelligence* **161**(1-2) (2005) 149–180
19. Petcu, A., Faltings, B.: DPOP: A scalable method for multiagent constraint optimization. In: Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI '05), Edinburgh, Scotland (Aug 2005) 266–271
20. Fudenberg, D., Levine, D.K.: Consistency and cautious fictitious play. *Journal of Economic Dynamics and Control* **19** (1995) 1065–1089
21. Crawford, V.P.: Adaptive dynamics in coordination games. *Econometrica* **63** (1995) 103–143
22. Hart, S., Mas-Colell, A.: A reinforcement procedure leading to correlated equilibrium. In Debreu, G., Neufeld, W., Trockel, W., eds.: *Economic Essays: A Festschrift for Werner Hildenbrand*. Springer, New York, NY (2001) 181–200
23. La Mura, P., Pearson, M.R.: 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)
24. Blum, A., Mansour, Y.: From external to internal regret. *Journal of Machine Learning Research* **8** (June 2007) 1307–1324
25. Pearce, J.P., Tambe, M.: Quality guarantees on k -optimal solutions for distributed constraint optimisation problems. In: Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI '07), Hyderabad, India (2007) 1446–1451
26. Anderson, S.P., de Palma, A., Thisse, J.F.: *Discrete Choice Theory of Product Differentiation*. MIT Press, Cambridge, MA (1992)
27. Singh, S., Jaakkola, T., Littman, M.L., Szepesvári, C.: Convergence results for single-step on-policy reinforcement-learning algorithms. *Machine Learning* **39** (2000) 287–308
28. Metropolis, N., Rosenbluth, A., Rosenbluth, M., Teller, A., Teller, E.: Equation of state calculations by fast computing machines. *Journal of Chemical Physics* **21** (1953) 1087–1092
29. Kirkpatrick, S., Gelatt, C.D., Vecchi, M.P.: Optimisation by simulated annealing. *Science* **220** (1983) 671–680
30. Berger, U.: Brown's original fictitious play. *Journal of Economic Theory* **135**(1) (2007) 572–578
31. Leslie, D.S., Collins, E.J.: Generalised weakened fictitious play. *Games and Economic Behavior* **56** (2006) 285–298
32. Monderer, D., Shapley, L.S.: Fictitious play property for games with identical interests. *Journal of Economic Theory* **68** (1996) 258–265
33. Chapman, A.C., Rogers, A., Jennings, N.R.: Benchmarking hybrid algorithms for distributed constraint optimisation games. In: Proceedings of the First International Workshop on Optimisation in Multi-Agent Systems (OptMas '08), Estoril, Portugal (2008)