A Distributed Algorithm for Optimising over Pure Strategy Nash Equilibria*

Archie C. Chapman
Electronics and Computer Science
University of Southampton, UK
acc@ecs.soton.ac.uk

Alessandro Farinelli
Computer Science Department
University of Verona, Italy
alessandro.farinelli@univr.it

Enrique Munoz de Cote, Alex Rogers and **Nicholas R. Jennings**
Electronics and Computer Science
University of Southampton, UK
{jemc,acr,nrj}@ecs.soton.ac.uk

Abstract

We develop an efficient algorithm for computing pure strategy Nash equilibria that satisfy various criteria (such as the utilitarian or Nash–Bernoulli social welfare functions) in games with sparse interaction structure. Our algorithm, called *Valued Nash Propagation* (VNP), integrates the *optimisation* problem of maximising a criterion with the *constraint satisfaction* problem of finding a game's equilibria to construct a criterion that defines a c–semiring. Given a suitably compact game structure, this criterion can be efficiently optimised using message–passing. To this end, we first show that VNP is complete in games whose interaction structure forms a hypertree. Then, we go on to provide theoretic and empirical results justifying its use on games with arbitrary structure; in particular, we show that it computes the optimum $>$82% of the time and otherwise selects an equilibrium that is always within 2% of the optimum on average.

1 Introduction

Game theory is increasingly being used as a modelling and design framework in artificial intelligence, particularly in the main field of multi–agent systems. Now, such systems are starting to contain large numbers of agents, producing three computational problems. The first is to find compact representations of games. In this work, we consider two compact graphical representations known as *graphical normal form* and *hypergraphical normal form* (Kearns, Littman, and Singh 2001; Gottlob, Greco, and Scarcello 2005; Papadimitriou and Roughgarden 2008), which can be exponentially more compact than the standard normal form if the agents' interaction structure is sufficiently sparse.

The second problem is to derive efficient methods for solving such games, typically in the form of a *Nash equilibrium*. In many multi–agent settings, a system designer particularly wants solutions that are pure strategy Nash equilibria (PSNE), because they imply a stable action profile and do not rely on utilities representing a cardinal ordering over outcomes, unlike mixed strategy equilibria. Consequently, we concentrate on computing such PSNE. To date, several methods have been suggested that exploit the compact representations described above, including the NashProp algorithm for graphical games (Kearns, Littman, and Singh 2001; Ortiz and Kearns 2003), Daskalakis and Papadimitriou's (2006) adaptation of the max–product algorithm that reduces a game to a Markov random field, and three algorithms that work by mapping the Nash equilibrium computation problem to a constraint satisfaction problem (CSP) — PureProp (Soni, Singh, and Wellman 2007), an algorithm by Vickrey and Koller (2002) and one by Gottlob, Greco and Scarcello (2005) (GGS in the remainder). [1]

The third challenge, in the presence of multiple equilibria, is to choose between them according to some criterion. This challenge is particularly relevant in design and/or control settings in which equilibrium is a necessary, but insufficient, condition for a solution, and yet, this problem has received the least attention in the algorithmic game theory literature. Kearns, Littman, and Singh (2001) note that NashProp can be modified to incorporate values ranking equilibria, and give some examples of permissible criteria. Elkind, Goldberg, and Goldberg (2007) derive a dynamic programming algorithm for selecting "good" equilibria according to similar criteria. Gottlob, Greco, and Scarcello (2005) show that finding a Pareto–dominant PSNE is computationally equivalent to finding an arbitrary PSNE. Finally, Greco and Scarcello (2004) characterise the complexity of the problem of optimising over the set of PSNE, and provide a centralised algorithm for the task. However, from the perspective of an agent systems designer looking for a control / optimisation method for real applications, what is missing from these investigations is the specification of a general–purpose distributed algorithm for these problems.

To address this shortcoming, we develop a general–purpose algorithm, called *Valued Nash Propagation* (VNP), for efficiently computing a PSNE that optimises many criteria (including the most commonly used ones) in games with bounded hypertree structure, whether they be in graphical

*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)

[1] Note that, because we consider only PSNE, we can focus on the topological and algebraic structure of the optimisation problems at hand, and not concern ourselves with techniques for finding mixed equilibria (c.f. (Kearns, Littman, and Singh 2001; Vickrey and Koller 2002; Soni, Singh, and Wellman 2007)).

or hypergraphical normal form. Specifically, we develop two versions of VNP, a complete algorithm for games with acyclic interaction structure, and an approximate algorithm for games with loopy topologies. These algorithms can optimise over PSNE using any equilibrium selection criterion that defines a valuation algebra on a *commutative* (c–) *semiring*. One of VNP's strengths is the diversity of selection criteria that it can optimise, which makes it suitable for various scenarios in which different criteria might be applied. Examples include the utilitarian, Nash–Bernoulli and egalitarian social welfare functions, threshold constraints on utility values, (e.g. minimum utility to a set of players), and the Pareto dominance and p–dominance equilibrium refinements (a generalisation of risk dominance). By so doing, this work generalises and unifies many of the above strands of research on computing and ranking equilibria in games with graphical interaction structure. Our work is inspired by general results regarding the *generalised distributive law* (GDL) family of algorithms (Aji and McEliece 2000; Kohlas and Wilson 2008), which includes max–product mentioned earlier, max–sum for distributed constraint optimisation and sum–product for graphical probability models (Farinelli et al. 2008; Wainwright, Jaakkola, and Willsky 2004). The GDL algorithms have been successfully applied to optimisation problems in agent systems with both tree and loopy topologies. They are complete for any problem that has an acyclic structure (either naturally or by generating a tree decomposition) and, furthermore, they run efficiently on problems with bounded tree–width. However, the GDL algorithms themselves cannot be used for equilibrium selection in games because they do not compute equilibria. In contrast, VNP explicitly considers best responses alongside an ordering over outcomes, and in so doing, interleaves equilibrium computation with the optimisation problem of ranking equilibria.

Like the GDL algorithms, VNP uses a two–phase message passing scheme. In the first phase, tables are exchanged between neighbouring agents indicating the value of each local joint strategy. This data structure is similar to that of NashProp and Elkind et al.'s algorithm, which are themselves closely related to max–sum (due to their common origins in dynamic programming), however, VNP deals explicitly with hyperedges, and, uniquely, can operate on games in graphical or hypergraphical normal form. In the second phase of VNP, strategy assignments are passed through a spanning tree, ensuring all agents agree on a particular solution, in a manner reminiscent of value propagation in dynamic programming. Specifically, we derive two versions of VNP; a complete algorithm for hypertree games, and an approximate algorithm for games with loopy interaction structure. Furthermore, in the absence of theoretic guarantees, we empirically validate the use of the second version on games with loopy interaction structure.

The paper progresses as follows: In the next section we review relevant background material on noncooperative games, introduce the graphical and hypergraphical normal form representations, and in Section 3 we discuss the criteria computable by VNP. Then, in Section 4, we describe the operation of VNP on games with sparse interaction structure, and prove forms of convergence for the cases of acyclic and loopy topologies. Following this, in the absence of theoretic guarantees, in Section 5 we empirically evaluate VNP in games with loopy topologies. Section 6 concludes.

2 Noncooperative Games

A noncooperative game in standard normal form is a tuple, $\Gamma = \langle N, \{A_i, u_i\}_{i \in N} \rangle$, consisting of a set of *agents* $N = \{1, \ldots, n\}$, and for each agent $i \in N$, a set of (pure) *strategies* or *actions*, A_i, with joint strategy space $A = \times_{i=1}^{N} A_i$, and a *utility function* $u_i : A \to \mathbb{R}$. We use the notation $a = \{a_i, a_{\nu_i}\}$, where ν_i is the set of i's *neighbours*, which in this representation is $N \setminus i$. An agent's goal is to maximise its payoff, and its *best response*, $B_i(a_{\nu_i})$, is the set of i's best strategies given its neighbours' strategies: $B_i(a_{\nu_i}) = \arg\max_{a_i \in A_i} u_i(a_i, a_{\nu_i})$. Stable points are characterised by the set of *Nash equilibria*, which are defined as those joint strategy profiles, a^*, in which no individual has an incentive to change its strategy:

$$u_i(a_i^*, a_{\nu_i}^*) - u_i(a_i, a_{\nu_i}^*) \geq 0 \quad \forall\, a_i \in A_i,\ \forall\, i \in N.$$

As such, the Nash equilibrium condition can be expressed as the sum of a set of indicator functions, $I\{a_i \in B_i(a_{\nu_i})\}$, each equal to 0 when an agent plays a best response and $-\infty$ otherwise:

$$\max[V(a)] = \max_{a \in A} \left[\sum_{i \in N} I\{a_i \in B_i(a_{\nu_i})\} \right] \qquad (1)$$

This criterion is equal to 0 if a is a PSNE and $-\infty$ elsewhere.

2.1 Compact Graphical Forms of Games

We begin with the requisite graph theory, before moving on to the graphical and hypergraphical normal form game representations. In more detail, an undirected *graph* $G = \langle N, E \rangle$ is composed of a set of nodes (agents), $N = \{1, \ldots, n\}$, connected by a set of edges, E. Building on this, a *hypergraph* $H = \langle N, E \rangle$ is composed of a set of nodes, N, and a set of hyperedges, E, in which $E \subseteq \mathcal{P}(N)$, i.e. a hyperedge $e \in E$ is a subset of the nodes. A graph is a hypergraph in which all hyperedges contain two nodes. Here we only consider simple (i.e. no duplicate edges or self–loops) connected hypergraphs.

Because of their algorithmic benefits, we are particularly interested in acyclic graphs and hypergraphs, or *trees* and *hypertrees*. Trees are oriented around a *root*, and have the property that each node (except for the root) has one *parent*. A node can have any number of *descendants*, and those without descendent are called *leaves*. Regarding hypertrees, we rely on classical acyclicity (c.f. α–*acyclicity* (Beeri et al. 1983; Gottlob, Greco, and Scarcello 2005)). This is defined in reference to a hypergraph's *vertex–hyperedge incidence graph*: a bipartite graph with one set of nodes containing the nodes of the hypergraph and the second containing the hyperedges. A hypergraph is acyclic if its vertex–hyperedge incidence graph is a tree (as in Fig 1), and the leaves of this graph are called the leaves of the hypergraph.

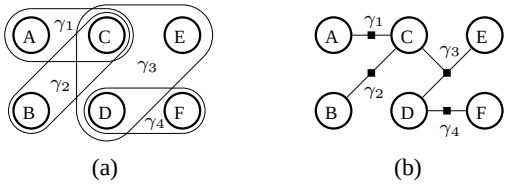

Figure 1: (a) An acyclic hypergraph and (b) its bipartite vertex–hyperedge incidence tree. Circles are agents, A to F, and hyperedges and squares are dependency hyperedges (graphical form) or local games (hypergraphical form), γ_1 to γ_4.

Graphical Normal Form Games in graphical normal form are represented by a graph on which each agent i is located at a node. An agent is connected to those with which it shares an undirected utility dependency, which make up its set of neighbours $\nu_i \subseteq N$. Its utility function is given by an array indexed by tuples from the set $\times_{j\in\{i,\nu_i\}}|A_j|$. If the game–graph is sufficiently sparse, this representation is exponentially more compact than the standard normal form (Kearns, Littman, and Singh 2001). We can extend this model to one in which cliques and loops are modelled by hyperedges. Specifically, we say that a hyperedge of a game in graphical form contains a subset of the agents, $\gamma \in \mathcal{P}(N)$, and the set of agents in γ is written N^γ. We denote the set of hyperedges containing i by Γ_i, the neighbours of i by $\nu_i = \cup_{\gamma\in\Gamma_i} N^\gamma \setminus i$, and i's neighbours in a particular hyperedge, γ, by $\nu_i^\gamma = N^\gamma \setminus i$. Finally, we say that a game in graphical form is tree–structured if its game graph forms a tree, and is hypertree–structured if the game hypergraph is acyclic (see Fig 1).

Hypergraphical Normal Form Hypergraphical normal form is used to represent noncooperative games that can be decomposed into several local games: $\Gamma = \{\gamma_1, \gamma_2, ..., \gamma_m\}$.[2] In this representation, nodes of a hypergraph correspond to the set of agents, while the hyperedges directly represent local games played between them (see Fig 1). Specifically, each hyperedge is a game in standard normal form: $\gamma = \langle N^g, \{A_i, u_i^g\}_{i\in N^g}\rangle$, where $N^g \subseteq N$ are the set of agents playing γ_g and $u_i^g : A^g \to \mathbb{R}$ is the payoff to i from its involvement in γ_g, where $A^g = \times_{j\in N^g} A_j$. Agents are usually involved in more than one local game, with the set of local games in which i is involved denoted Γ_i. For each agent, a_i represents i's global strategy, which is the same in each local game. Agent i's neighbours are the agents that it shares at least one local game with, defined as $\nu_i = \cup_{\gamma_g\in\Gamma_i} N^g \setminus i$, and i's neighbours in γ_g are $\nu_i^g = N^g \setminus i$. Agent i's payoff for an outcome is an aggregation of its payoffs from each $\gamma_g \in \Gamma_i$, typically the sum of payoffs from each: $u_i(a_i, a_{\nu_i}) = \sum_{\gamma_g\in\Gamma_i} u_i^g(a_i, a_{\nu_i^g})$. If the agents' utilities from different interactions separate into many local games, then the entire representation can be exponentially smaller than the standard normal form (Papadimitriou and Roughgarden 2008). Many computational situations arising in control settings naturally possess hypergraphical structure, particularly those in which neighbours are specified with reference to a physical domain in which interactions are dependent on some degree of physical proximity (e.g. mobile sensors, UAVs, or job scheduling problems). As for the graphical normal form, an acyclic hypergraphical form game is one whose local game hypergraph is acyclic (as shown in Fig 1).

[2]We use the same notation for hyperedges in both the graphical and hypergraphical forms because, from an algorithmic perspective, the two types of hyperedges are treated the same way, even though they do not necessarily represent identical relationships between the agents.

3 Equilibrium Selection Criteria

In general, any criterion that defines a c–semiring can be computed by VNP, and many selection criteria have this form. In more detail, a *commutative semiring* (c–semiring) is a tuple $\langle K, \oplus, \otimes\rangle$ defining a *carrier*, K, and two binary operations $\oplus$ and $\otimes$. These operations possess identities in K and satisfy the following two rules: (i) $\otimes$ distributes over $\oplus$ (i.e. $x \otimes (y \oplus z) = x \otimes y \oplus x \otimes z$) and (ii) both are commutative (e.g. $x \otimes y = y \otimes x$) for any $x, y, z \in K$.

One commonly–employed equilibrium selection criterion is the *utilitarian* social welfare function (SWF), which selects the PSNE that maximises the sum of the agents' utilities. We can construct a criterion for this problem by augmenting the PSNE criterion (Eq 1) with the utilitarian SWF:

$$\max_{a\in A}[V(a)] = \max_{a\in A}\left[\sum_{i\in N}\left(I\{a_i \in B_i(a_{\nu_i})\} + u_i(a_i, a_{\nu_i})\right)\right] \quad (2)$$

This criterion returns $-\infty$ if a profile that is not a PSNE is played, and the sum of the agents' utilities when one is played. Consequently, it defines a valuation algebra on the tropical semiring, $\langle\{\mathbb{R}\cup-\infty\}, \max, +\rangle$. Furthermore, the image set of the indicator functions, $\{0, -\infty\}$, are the idempotent elements of $\{\mathbb{R}\cup-\infty\}$, and the value returned if a_i is not a best response is absorbing under addition. In general, in terms of the abstract operators defined earlier, max and $+$ correspond to $\oplus$ and $\otimes$, and the indicator function returns a value that either annihilates (and absorbs under $\otimes$) the value in that element of the table if it is not a best response, or leaves the value unchanged if it is a best response. Any selection criteria with this form can be optimised using VNP; two further examples are the *egalitarian* SWF and *Nash–Bernoulli product* (Elkind et al., 2007, give more examples of criteria fitting this form). The former ranks equilibria according to the minimum utility received by any agent, and employs an indicator function that returns ∞ if a_i is a best response and $-\infty$ if it is not. The latter aggregates individual utilities by multiplication, with an indicator function returning 1 for a best response and zero otherwise.

4 The VNP Algorithm

VNP operates using a two–phase message–passing sequence, called *table–passing* and *assignment–passing*. As message–passing on loopy topologies is inherently more complicated, in what follows we distinguish between the tree and loopy hypergraph cases. Specifically, in Section 4.1

we describe the table–passing phase, which is the key component of VNP, and prove a form of convergence for each case. Then, in Section 4.2 we describe the assignment passing phase, and prove that a single criterion–maximising outcome is selected by the agents on hypertrees, and that if the table–passing phase converges, a high value PSNE is selected on loopy hypergraphs. For reference, pseudocode of VNP's phases on hypertrees is given in Fig 2, and an example of its operation is given in Chapman et al. (2010).

4.1 The Table–Passing Phase

In this phase, agents exchange tables with their neighbours in each hyperedge. These tables contain values which succinctly represent the value of the ranking of each joint strategy of the hyperedge's agents. Although VNP's message–passing schedule and termination condition for games on hypertrees differ from that for loopy hypergraphs, the data structures and the operations used to construct the values in the messages are common to both. As such, we now detail the common elements, and then discuss the respective message–passing schedules and forms of convergence for hypertrees and loopy hypergraphs independently.

The messages exchanged in the table–passing phase are an array, indexed by the joint strategy space of the agents in the hyperedge common to both sender and recipient. Specifically, i passes to its neighbour j in hyperedge γ_g an array $\mathcal{T}_{i\to j}$ with $\prod_{k\in N^g}|A_k|$ entries, indexed by an ordered $|N^g|$–tuple representing a joint strategy of the agents in γ_g. An element of a message is denoted $\mathcal{T}_{i\to j}(a_i, a_{\nu_i^g})$. Agents store the messages they receive for use in the second phase.

Using the notation of Eq 2, elements of the message i sends to j are computed by:

$$\mathcal{T}_{i\to j}(a_i, a_{\nu_i^g}) = \max_{a_{\nu_i^{-g}}} \Bigg[I\{a_i \in B_i(a_{\nu_i^g}, a_{\nu_i^{-g}})\} + u_i(a_i, a_{\nu_i^g}, a_{\nu_i^{-g}}) \sum_{\gamma_h \in \Gamma_i \setminus \gamma_g} \sum_{k \in \nu_i^h} \mathcal{T}_{k\to i}(a_i, a_k, a_{\nu_k^h \setminus i}) \Bigg] \tag{3}$$

where $\nu_i^{-g} = \nu_i \setminus \nu_i^g$. The operations above perform two processes. First, for a fixed joint strategy $(a_i, a_{\nu_i^g})$ in γ_g, the summations *combine* the value of the joint strategy of the agents in all of the hyperedges containing i except γ_g (i.e. $\Gamma_i \setminus \gamma_g$) to give rankings to each joint strategy of i's neighbours outside of γ_g. In this combination, the indicator function sets the entries to the annihilating element of the semiring if the joint strategy is not in equilibrium with a_i. Second, the maximisation then *summarises* the information regarding $a_{\nu_i^{-g}}$ and projects the information onto the fixed joint strategy in γ_g. If there is no equilibrium associated with a_i, then the output is the annihilating element. Now, to complete the maximisation component, an agent needs to sum messages regarding strategies of different agents. To do this, the agent *extends* each incoming message to the joint strategy space of all incoming messages, by combining the relevant message entries to give a value for the respective joint strategy. As an example, consider Fig 1, in which C would extend messages it received from A and B to $|A_A| \times |A_B| \times |A_C|$ and then perform the maximisation.

When the termination condition for the table–passing phase is met, agent i uses the messages it received from its neighbours to construct the following function:

$$V_i(a_i, a_{\nu_i}) = I\{a_i \in B_i(a_{\nu_i})\} + u_i(a_i, a_{\nu_i}) \sum_{\gamma_g \in \Gamma_i} \sum_{j \in \nu_i^g} \mathcal{T}_{j\to i}(a_i, a_{\nu_i^g}) \tag{4}$$

Next, we show that the above result is equal to the equilibrium selection criterion (Eq 2) in games with hypertree interaction structure (Theorem 1), while with loopy structure, the values of non-PSNE local profiles converge to the annihilating element of the semiring (Theorem 2).

On Hypertrees Table–passing begins with messages sent from the leaves of the hypergraph. Each internal node then computes and sends messages to its neighbours in a particular local game γ_g once it has received messages from all of its neighbours outside γ_g. After it receives messages from all of its neighbours, i constructs the function given in Eq 4. By computing the maximum value of Eq 4 (i.e. over $\{a_i, a_{\nu_i}\}$), i finds a local configuration that maximises Eq 2.

Theorem 1 *The table–passing phase of VNP produces the value of the relevant c–semiring criterion for each local pure strategy profile in a hypertree–structured game.*

The proof is by induction, and is available in Chapman et al. (2010). If the solution is unique or doesn't exist, then the table's values show this unambiguously, and each agent selects its strategy without requiring an assignment–passing phase. However, if more than one solution exists, the agents coordinate on a solution using assignment–passing (discussed in Section 4.2).

On Loopy Hypergraphs Even though the table–passing phase of VNP is only *provably* optimal on acyclic hypergraphs, we may, want to deploy it on loopy hypergraphs. Fortunately, max–sum and NashProp perform well on such topologies (Farinelli et al. 2008; Ortiz and Kearns 2003), and because of its similarity to these algorithms, we conjecture that VNP will also perform well on loopy hypergraphs (see Section 5 for more details). However, this necessitates changes to the message schedule and termination condition of the table–passing phase as described for hypertrees.

The message–passing schedule used on arbitrary topologies is a "flood" schedule, whereby every agent sends a message at each time step (this is not the only option, but has been used to good effect in Farinelli et al. (2008) and Ortiz and Kearns (2003)). The agents begin by initialising all elements of their "stored" messages to 0 (or the identity of $\otimes$ in the relevant semiring). Then, at each time step t, they simultaneously compute messages according to Eq 3, and send them to their neighbours.

Now, loops in the hypergraphs prevent us from explicitly calculating the value of the criterion. Nonetheless, we can put a weaker guarantee on the values in the messages passed by VNP in loopy graphs, and consequently the value of each

Function vnpTablePassing(Γ_i)
for each hyperedge $g : \Gamma_i$,
  if values $\mathcal{T}_{j\to i}$ have been received from all neighbours outside g, $j \in \nu_i^{-g}$,
    compute $\mathcal{T}_{i\to j}$ for neighbours in g, $j \in \nu_i^g$;
    send $\mathcal{T}_{i\to j}$ to neighbours in g, $j \in \nu_i^g$;
  end if
end for

Function vnpAssignmentPassing(Γ_i, $\mathcal{T}_{j\to i}$ $\forall j \in \nu_i$)
if an assignment $\mathcal{S}_{j\to i}$ has been received from any neighbour, $j \in \nu_i^g$ $\forall g \in \Gamma_i$,
  for each hyperedge $g : \Gamma_i$,
    if no neighbour $k \in g$ has sent a message $\mathcal{S}_{k\to i}$,
      compute $\mathcal{S}_{i\to k}$ for neighbours in g, $k \in \nu_i^g$;
      send $\mathcal{S}_{i\to k}$ to neighbours in g, $k \in \nu_i^g$;
    end if
  end for
end if

Figure 2: Pseudocode of each phase of VNP on a hypertree.

agent's local version of the criterion (Eq 4). Let $\mathcal{T}_{i\to j}^t$ be the message passed from i to j at time t. Call a message entry *eliminated* if it equals $-\infty$ (or the annihilating element of the semiring in use) and *valued* otherwise.

Theorem 2 *For VNP on a game with arbitrary topology, as $t \to \infty$, a message entry is valued if and only if it corresponds to local strategy profile that is a PSNE.*

The proof is sketched as follows:[3] First, for sufficiency, let a^* be a PSNE profile. Then for all $t \geq 0$, all message elements corresponding to the subset of a^* in γ_g, $\mathcal{T}_{i\to j}^t(a_i^*, a_{\nu_i^g}^*)$, are valued. This can be shown to be true by induction: For $t = 0$ it is true by the initialisation of the stored messages, and then for any $t > 0$, any entry of a message sent to j corresponding to a^* is valued, and j itself will never eliminate that entry, so it is true for all t, and therefore true for the limit messages. Thus we have proved the sufficient condition. Second, to prove necessity, note that once an entry in a stored message is eliminated, it cannot become valued again. Consequently, because any global profile that contains at least one action that is not a best response is eliminated at some $t < \infty$, any valued entry in a stored message that does not correspond to a PSNE profile is also eliminated.

In other words, in the limit, Eq 4 is only valued for local profiles corresponding to a PSNE, and, therefore, when the assignment–passing phase selects an outcome based on Eq 4, it is always a PSNE. Furthermore, if no PSNE solution exists, then the algorithm will report this fact because all the entries in all the messages will converge to the annihilating element. However, because the values of the (non–eliminated) entries in the tables do not necessarily converge to the true values of Eq 2, the equilibrium selected by the agent may not maximise the criterion. Nonetheless, we expect that the equilibrium selected should still do very well by the criterion, and test this hypothesis in Section 5, where we run VNP on a batch of loopy hypergraphical games.

Although we have proven a form of convergence for table–passing on loopy graphs, it is not possible to precisely define a termination condition for this phase. A partial condition is that table–passing should terminate if all of the messages being passed converge. However, this cannot be known beforehand, so, in its absence, this phase should allow sufficient time for the values in the messages passed by the agents to converge, if indeed they are going to converge (details of our implementation are presented in Section 5).

4.2 The Assignment–Passing Phase

Once the termination condition for the table–passing phase has been satisfied, the function in Eq 4 is computed and each agent uses it to decide on a strategy. If a unique solution (or the non–existence of one) is not evident at the end of the table–passing phase, then the agents coordinate on a single solution using an assignment–passing phase. As before, we need to use different approaches on hypertrees and loopy hypergraphs, and the two cases are discussed separately below.

On Hypertrees To begin, a spanning tree is constructed, emanating from an arbitrarily selected root node, down which parents pass messages containing local strategy profiles to their descendent This tree is built by first connecting the root to all of its neighbours. A particular agent, i, which interacts with the root through γ_g, is either a leaf or an internal node involved in other hyperedges. If it is an internal node, then it is the only agent that appears in both γ_g and any of its other hyperedges (by the definition of acyclicity). As such, the spanning tree is extended by connecting i to all of its neighbours except those in γ_g, and so on.

Given this spanning tree, the root randomly selects a local strategy profile corresponding to one maximum of its version of Eq 4. For the root node, the action is given by $\mathrm{argmax}_{a_i, a_{\nu_i}}[V_i(a_i, a_{nu_i})]$. It then sends messages, $\mathcal{S}_{r\to i}$ to its neighbours in each hyperedge, $i \in N^{\gamma_g}$, containing the local strategy, a_{N^g}, for γ_g. In other words, the root selects a complete strategy profile for each of its hyperedges and directs each neighbour to play its element of this profile. As for the table–passing computations, this operation is polynomial in the size of the local joint action space of a player in both the graphical and hypergraphical representations of the game. For an internal i, a_i and some of the actions in a_{ν_i} will be assigned by the agent's parent node, so the argmax operation is carried out over only the unassigned variables In hyperedges with three or more agents, this is necessary to avoid mis–coordination between two of the root's neighbours, because more than one optimal joint strategy could be associated with the root's strategy. Then i selects a joint strategy for all of its hyperedges *except* γ_g that both maximises its version of Eq 4 and contains the strategy assigned to it by the root, choosing randomly between equivalent strategies. It sends $\mathcal{S}_{i\to j}$ strategy assignments to all of its unassigned neighbours j. This process continues until each leaf of the spanning tree is assigned a strategy by its parent. At this point, all agents have been assigned a strategy and the algorithm terminates. Conceivably, the span-

[3] Our proof follows a similar argument to that of Ortiz and Kearns (2003) for the convergence of messages in the NashProp algorithm in loopy graphical games.

ning tree could be constructed and the strategies propagated as part of one process.

Lemma 1 *The assignment–passing phase of VNP for hypertrees assigns each agent a strategy corresponding to a single criterion–maximising PSNE in a hypertree–structured game.*

The proof is omitted for brevity, but follows the same argument for the max–product algorithm presented in Wainwright, Jaakkola, and Willsky (2004). However it should be clear from the above that, by induction, the root's choice propagates through the width–first spanning tree without any mis–coordination of agents' strategies. Combining Theorem 1 and Lemma 1 gives us the following result.

Theorem 3 *In acyclic graphical and hypergraphical games, for any selection criterion that defines a valuation algebra on a c–semiring, VNP computes an optimal PSNE.*

Additionally, if the number of neighbours and the number of hyperedges each agent has is bounded, VNP is polynomial in the total number of agents, because each local computation is polynomially bound.[4] Finally, on a hypertree, the number of messages exchanged in order to carry out both phases of computation in VNP is at most $r + s$ steps, where r is the diameter of the game hypertree and s is the length of the longest path from root to leaf in the assignment spanning tree (if an assignment–passing phase is required).

On Loopy Hypergraphs On loopy hypergraphs, if more than one solution exists and/or table–passing does not converge, then the agents must use a more sophisticated strategy assignment procedure than on acyclic hypergraphs. Again, the first step is to build a spanning tree, in the same manner as for acyclic topologies, with the additional condition that an agent only links to its unconnected neighbours.

However, the presence of loops may mean that neighbours in the game may have different parents in the spanning tree. Then, in the case of multiple optimal solutions, if the procedure for hypertrees described earlier is followed, these agents may be assigned conflicting strategies by different branches of the spanning tree. To avoid this, we use a backtracking search over the spanning tree. By this approach, a partial solution is constructed as described earlier, beginning with the root node optimising its version of Eq 4 and sending its neighbours the corresponding local strategy. This process continues until any agent identifies a conflict in the strategies assigned to it or one of its neighbours (i.e. a non–equilibrium profile is identified). If this occurs, the search backtracks to a point where the conflict can be resolved, and then begins propagating a solution again. However, the search process can take an exponentially long time to complete in the case of hypergraphs with many cycles.

Overall, the VNP algorithm on games with arbitrary structure will never return a solution that is not a PSNE, and we expect it to provide solutions that approximately optimise the criterion in question. To justify this claim, we now provide an empirical evaluation of VNP on loopy games.

[4] Gottlob, Greco, and Scarcello (2005) prove this for arbitrary and Pareto PSNE computation on α–acyclic hypergraphs. In contrast, when ranking equilibria by a c–semiring criterion, the correctness of the computation only holds for *classical* acyclicity.

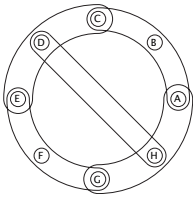

Figure 3: Example of the chordal topology used in the experiments.

5 Evaluation of VNP on Loopy Hypergraphs

The purpose of this section is to demonstrate, in the absence of theoretic guarantees, the efficacy of VNP on games with sparse but cyclic topologies. Specifically, we use a set of hypergraphical games with *chordal* topology, following Ortiz and Kearns (2003). These games have sparse–enough topology to demonstrate how VNP exploits structure (because each agent has a limited number of neighbours), but at the same time are sufficiently cyclic to make the exact computation of a criterion infeasible for large problems. The games are constructed using a generic topology, shown in Fig 3, comprising (i) a ring of 3–player, 2–action local games, such that every second agent is contained on two hyperedges and the games form a single cycle, and (ii) chord games, linking pairs of agents that are only in one ring game (which limits the number of hyperedges any one agent is on to two). The payoffs in the local games are constructed such that they possess several pure equilibria, however, the ranking over these equilibria varies according to randomly chosen payoffs in the chord games. More details of the experiment generating process are available in Chapman et al. (2010).

We compared VNP for loopy topologies to an optimal 'brute force' version of VNP that does not exploit any graphical structure, in which each agent constructs a joint message to every other in the system and so maximise Eq 2 directly. We did not benchmark against existing approaches because ours is the only algorithm that optimises over any c–semiring criterion; indeed we could have cast VNP as an extension to either NashProp or GGS for the purpose of optimising over equilibria in games with sparse hypergraphical interaction structure. As such, our experiments only aim to prove the efficacy of VNP.

We ran 100 iterations for each size game. We say an algorithm has converged when the optimal strategies of the agents have not changed for 10 time steps, and recorded the convergence time as the last of these steps. We also recorded the average processing time the algorithm took to converge[5], the proportion of runs that converge to an optimum and, for the remaining suboptimal solutions, the average ratio of the optimal solution's value over the suboptimal solution.

The results of these experiments are reported in Fig 1. Regarding the quality of the solutions generated, they show that VNP computes the optimal solution with a very high

[5] Experiments were carried out on a 3.2GHz desktop PC running Java, with all processes run in sequence. As such, processing times were recorded as a relative measure only.

Agents	Conv Steps	Time (ms)	% Optimal	Opt/Subopt
4	15.4 (0.32)	64 (5.8)	94% (4.8)	1.021 (0.0033)
8	17.6 (0.52)	163 (7.2)	91% (5.8)	1.010 (0.0025)
12	19.1 (0.69)	273 (10.7)	87% (6.8)	1.014 (0.0014)
16	20.1 (0.83)	428 (17.1)	84% (7.4)	1.011 (0.0018)
20	21.1 (0.83)	554 (21.6)	82% (7.8)	1.007 (0.0012)
24	21.9 (1.30)	623 (29.9)	87% (6.8)	1.013 (0.0017)

Table 1: Message steps to converge, time to converge, proportion converging to an optimal solution and the average ratio of the optimum over the value of suboptimal solutions (standard errors in brackets).

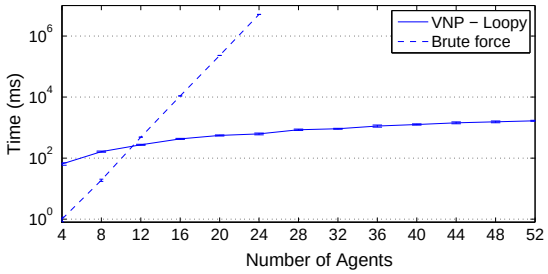

Figure 4: Convergence time (ms) versus number of agents, comparing VNP for loopy topologies to a brute force version of VNP that does not exploit interaction structure.

frequency (always greater than 82% of runs). Furthermore, when a sub–optimal equilibrium is selected, its value is typically very close to that of the optimum (always within 2%). In terms of the computation time required to generate a solution, the number of message–passing steps to find an equilibrium increases linearly with the number of agents. Furthermore, the actual computation time grows at a polynomial rate, which is in an artefact of the chordal topology with bounded number of neighbours we run the experiments on.[6] In order to better demonstrate the scalability of VNP if each agent's number of neighbours is bounded, in Fig 4 we plot VNP's computing time as the number of agents increases, compared to the brute force version. It shows that VNP's processing time increases at a rate that is orders of magnitude less than the brute force algorithm. Indeed, the processing time of the brute force approach follows an almost perfect exponential relationship with the number of agents in the problem, while VNP follows a polynomial relationship. Based on the theoretical properties of VNP discussed earlier (at the end of Section 4.2), we expect that these scaling results will genearlise in a similar fashion to problems with greater local neighbourhood size (i.e. the polynomial degree of the computation time should grow linearly with increases in the local neighbourhood size), and should also be invariant to the number or size of loops in the problem.

6 Conclusions

We have developed the VNP algorithm for computing PSNE that optimise various social welfare criteria in games with bounded hypergraphical structure. We have shown how to integrate the optimisation problem of finding the maximal element of a valuation algebra with the constraint satisfaction problem of finding a game's PSNE. We have also demonstrated that if the agents in a hypergraphical game each have a bounded number of neighbours and the game hypergraph has bounded hypertree–width, then the problem can be solved efficiently by VNP. In so doing, we have completely characterised the types of problems that can be addressed by algorithms of VNP's form, and provided two variants of VNP for different topologies. To round out research on VNP, future work will focus on developing a complete version for loopy graphs that operates without first forming a junction tree and deriving analogous convergence results for VNP to those regarding specific topologies and attenuated messages for the GDL algorithms.

[6]Note that NashProp would compute an arbitrary PSNE in a similar number of steps, however, the computation time would likely be less because it does not use floating point operations.

References

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

Beeri, C.; Fagin, R.; Maier, D.; and Yannakakis, M. 1983. On the desirability of acyclic database schemes. *J. ACM* 30(3):479–513.

Chapman, A. C.; Farinelli, A.; de Cote, E. M.; Rogers, A.; and Jennings, N. R. 2010. Appendix to *A Distributed Algorithm for Optimising over Pure Strategy Nash Equilibria* . Available at `http://www.ecs.soton.ac.uk/people/acc/publications`.

Daskalakis, C., and Papadimitriou, C. H. 2006. Computing pure Nash equilibria via Markov random fields. In *ACM EC '06*.

Elkind, E.; Goldberg, L. A.; and Goldberg, P. W. 2007. Computing good Nash equilibria in graphical games. In *EC '07*, 162–171.

Farinelli, A.; Rogers, A.; Petcu, A.; and Jennings, N. R. 2008. Decentralised coordination of low-power embedded devices using the max-sum algorithm. In *AAMAS '08*, 639–646.

Gottlob, G.; Greco, G.; and Scarcello, F. 2005. Pure Nash equilibria: Hard and easy games. *JAIR* 24:357–406.

Greco, G., and Scarcello, F. 2004. Constrained pure Nash equilibria in graphical games. In *ECAI '04*.

Kearns, M.; Littman, M. L.; and Singh, S. 2001. Graphical models for game theory. In *UAI '01*, 253–260.

Kohlas, J., and Wilson, N. 2008. Semiring induced valuation algebras: Exact and approximate local computation algorithms. *Art. Int.* 172(11):1360–1399.

Ortiz, L. E., and Kearns, M. J. 2003. Nash propagation for loopy graphical games. In Becker, S.; Thrun, S.; and Obermayer, K., eds., *NIPS*, volume 15, 793–800.

Papadimitriou, C. H., and Roughgarden, T. 2008. Computing correlated equilibria in multi–player games. *J. ACM* 55(3):14–29.

Soni, V.; Singh, S.; and Wellman, M. P. 2007. Constraint satisfaction algorithms for graphical games. In *AAMAS '07*, 423–430.

Vickrey, D., and Koller, D. 2002. Multi-agent algorithms for solving graphical games. In *AAAI '02*, 345–351.

Wainwright, M.; Jaakkola, T.; and Willsky, A. 2004. Tree consistency and bounds on the performance of the max-product algorithm and its generalizations. *Stat. and Comp.* 14(2):143–166.