Solving Fuzzy Distributed CSPs: An Approach with Naming Games*,**

Stefano Bistarelli[1,2], Giorgio Gosti[3], and Francesco Santini[1,4]

[1] Dipartimento di Matematica e Informatica, Università di Perugia
{bista,francesco.santini}@dmi.unipg.it
[2] Istituto di Informatica e Telematica (CNR), Pisa, Italy
stefano.bistarelli@iit.cnr.it
[3] Institute for Mathematical Behavioral Sciences, University of California, Irvine, USA
ggosti@uci.edu
[4] Contraintes, INRIA - Rocquencourt, France
francesco.santini@inria.fr

Abstract. Constraint Satisfaction Problems (CSPs) are the formalization of a large range of problems that emerge from computer science. The solving methodology described here is based on *Naming Games* (*NGs*). NGs were introduced to represent N agents that have to bootstrap an agreement on a name to give to an object (i.e., a word). In this paper we focus on solving both Fuzzy NGs and Fuzzy Distributed CSPs (Fuzzy DCSPs) with an algorithm inspired by NGs. In this framework, each proposed solution is associated with a preference represented as a fuzzy score. We want the agents to find the solution, which is associated with the highest preference value among all solutions. The two main features that distinguish this methodology from classical Fuzzy DCSPs algorithms are that *i)* the system can react to small instance changes, and *ii)* the fact the algorithm does not require a pre-agreed agent/variable ordering.

1 Introduction

In this paper we present a distributed algorithm to solve *Fuzzy Distributed Constraint Satisfaction Problems* (*Fuzzy DCSPs*) [14,18,11,12,17] that comes from a generalization of the *Naming Game* paradigm (*NG*) [15,1,13,10].

In Fuzzy DCSPs algorithms, the aim is to design a distributed architecture of processors, or more generally a group of agents, which cooperate to solve a particular Fuzzy DCSP instantiation. In the framework presented here, we see the Fuzzy DCSP solution search as a dynamic system, and we set the stable

* This work was carried out during the tenure of the ERCIM "Alain Bensoussan" Fellowship Programme, which is supported by the Marie Curie Co-funding of Regional, National and International Programmes (COFUND) of the European Commission.
** Research partially supported by MIUR PRIN 2010-2011 2010FP79LR project: "Logical Methods of Information Management".

M. Baldoni et al. (Eds.): DALT 2012, LNAI 7784, pp. 116–135, 2013.

states of the system as the solutions to our Fuzzy DCSP. To do this we design each agent so that it moves towards a stable local state. This system may be called "self-stabilizing" whenever the global stable state is obtained through the reinforcement of the local stable states [7]. The system settles to a global stable state when all agents are in stable local state, When the system finds this global stable state the DCSP instantiation is solved. A protocol designed in this way is resistant to damage and external threats, since it can react to small changes in the original problem instance. Moreover, in our approach all agents have the same probability to reveal private information, and for this reason such algorithm is unbiased (i.e., "fair") with respect to privacy.

The NG paradigm defines a set of problems where a number of agents bootstrap a commonly agreed name for one or more objects [15,1,13,10]. In this paper we discuss a NG generalization in which agents have individual fuzzy preferences over words. This is a straightforward generalization of the NG, because this paradigm naturally models the endogenous agents preferences and the attitudes towards a given naming system. These preferences may be driven by pragmatic or rational reasons: same words may be associated to different objects, same words may be too long or too complex, or may be easy to confuse and, therefore, less suitable as a solution for name assignments.

In Sec. 4, we define Fuzzy NG which are a generalization of the NG that introduces agent preferences. To model agents preferences we associate individual fuzzy levels with each word in the agents domain. In this way, the new game may be interpreted as an optimization problem. The Fuzzy NG we obtain can be seen as a particular instance of a Fuzzy DCSP with fuzzy unary constraints and crisp binary constraints which impose that the possible solutions are the ones in which all the agents connected by a communication link share the same word as a naming convention. Then we extend the works in [3] and [4] in order to consider agent preferences. Within this set of candidate solutions, the real solutions are the ones that optimize the overall preference for the agreed name. All the agents agree on the same word, which is the best possible according to the composition of the preferences of all the participating agents. Since we use fuzzy preferences, values are in the interval $[0, 1]$, they are aggregated with the min operator, and to optimize means to find the word with the maximum preference possible (with the max operator).

The algorithm is defined through an asymmetric interaction among agents, in which one peer is the "speaker" and the other involved agents are called "listeners". To let this interaction occur, our algorithm uses a central scheduler that randomly draws a speaker at each round. This may be interpreted as a "central blind orchestrator" scheme, anyhow this central scheduler has no information on the DCSP instance, and has no pre-determined agent/variable ordering: therefore, it preserves the privacy of the agents.

In Sec. 5 we explain how the algorithm in Sec. 4.1 can be extended to solve a generic instance of a Fuzzy DCSP, that is a DCSP problem where both unary and binary constraints are associated with a fuzzy preference. Fuzzy DSCPs can

be used to deal with resource allocation, collaborative scheduling and distributed negotiation [11].

In summary, the main contributions of this paper are two. First, we discuss how individual preferences can be modeled in the NG with the use of the Fuzzy NG, and how we can use a distributed algorithm to solve this problem. Second, we discuss ho a similar algorithm can be used to solve more the more general class of Fuzzy DCSPs.

The paper extends preliminary work in [5], by refining the distributed algorithm and sketching a sample execution of the algorithm, to better understand its functioning. The paper is organized as follows: in Sec. 2 we respectively present the background on Fuzzy DSCPs and NGs, while Sec. 3 summarizes the related work. Section 4 presents an algorithm that solves Fuzzy NGs. Section 5 shows how to extend the algorithm in Sec. 4 in order to solve generic Fuzzy DCSPs. Then, in Sec. 6 we show a simple example on how the algorithm in Sec. 5 works, and Sec. 7 presents the tests and the results for the Fuzzy NG algorithm. Finally, in Sec. 8 we draw our conclusions and explain our future work.

2 Background

2.1 Distributed Constraint Satisfaction Problem (DCSP)

A classical constraint can be seen as the set of value combinations for the variables in its scope that satisfy the constraint. In the fuzzy framework, a constraint is no longer a set, but rather a fuzzy set [14]. This means that, for each assignment of values to its variables, we do not have to say whether it belongs to the set or not, but how much it does so. In other words, we need to use a graded notion of membership. This allows us to represent the fact that a combination of values for the constraint variables is partially permitted. A Fuzzy CSP is defined as a triple $P = \langle X, D, C \rangle$, where X is the set of variables and D is the set of corresponding variable domains (we suppose a single domain for all the variables). C is a set of fuzzy constraints. A fuzzy constraint is defined by a function c_V on a sequence of variables V, which is called the *scope* (or *support*) of the constraint, that is the set of variables on which the constraint is defined on.

$$c_V : \prod_{x_i \in V} D_i \to [0, 1]$$

The function c_V indicates to what extent an assignment of the variables in V satisfies the constraint [14]. In fuzzy constraints, 1 usually corresponds to the best preference, and 0 to the worst preference value. The combination $c_V \otimes c_W$ of two fuzzy constraints c_V and c_W is a new fuzzy constraint $c_{V \cup W}$ defined as

$$c_{V \cup W}(\eta) = min(c_V(\eta), c_W(\eta))$$

where η is a complete assignment of the variables in the problem, i.e., an assignment of the variables in X:

$$\eta \in \prod_{x_i \in X} D_i$$

If $c_1\eta > c_2\eta$ (e.g., $c_1\eta = 0.8$ and $c_2\eta = 0.4$), it means that the assignment η satisfies c_1 better than c_2. In the following of the paper we will use the expression $c\eta[x_i := d]$ to denote a constraint assignment in which variable $x_i \in X$ takes the value $d \in D$.

We can now define the preference of the complete set C of constants in the problem, by performing a combination of all the fuzzy constraints. Given any complete assignment η we have

$$(\bigotimes_{c_V \in C} c_V)(\eta) = \min_{c_V \in C} c_V(\eta)$$

Thus, the optimal solutions of a fuzzy CSP are the complete assignments whose satisfaction degree is maximum over all the complete assignments, that is,

$$OptSol(P) = \{\eta \mid \max_{\eta} \min_{c_V \in C} c_V(\eta)\}$$

In the description of our algorithm in Sec. 4 we will also need a definition of *projection* for fuzzy constraints: given a fuzzy constraint $c_V \in \mathcal{C}$ and a variable $v \in V$, the *projection* [2] of c_V over v, written as $c_V \Downarrow_v$, is a fuzzy constraint c' such that $c'\eta = \max(c\eta[x_1 := d_1] \dots [x_k := d_k])$, where $d_1 \dots d_k \in D$ and $x_1 \dots x_k \in (V \backslash \{v\})$. For instance, if $V = \{v\}$, then $c_V \Downarrow_v = c_V$. Informally, projecting means to eliminate the influence of all the variables $V \backslash \{v\}$ over a constraint (i.e., to remove the variables in $V \backslash \{v\}$ from its scope), by considering the assignment that maximises the preference of c.

In DCSPs [18,14], the main difference to a classical CSP is that each variable is controlled by a corresponding agent, meaning that this agent sets the variable's value. Formally, a DCSP is a tuple $\langle X, D, C, A \rangle$, i.e., a CSP with a set A of n agents. We suppose the number of variables m to be greater/equal than the number of agents n, i.e., $m \geq n$. When an agent controls more than one variable, this can be modeled by a single variable whose values are the combinations of values of the original variable. It is further assumed that an agent knows the domain of its variable and all the constraints involving its variable, and that it can reliably communicate with all the other agents which share the same constraints. The main challenge is to develop distributed algorithms that solve the CSP by exchanging messages among the agents. Fuzzy DCSPs features both all the features described in this section, i.e., fuzziness and distributivity.

2.2 Introduction to Naming Games

The NG model [15,1,13,10] describes a set of problems in which a number of agents bootstrap a commonly agreed name for one or more objects.

The game is played by a population of n agents which play pairwise interactions in order to negotiate conventions, that is associations between forms and meanings, and it is able to describe the emergence of a global consensus among them. For the sake of simplicity this model does not take into account the possibility of homonymy, so that all meanings are independent and one can work with only one of them, without loss of generality. An example of such a game is a population that has to reach a consensus on the name (i.e., the form) to assign to an object (i.e., the meaning), by exploiting local interactions only. However, the same model is appropriate to address all those situations in which negotiation rules a decision process (e.g., opinion dynamics) [1].

Each NG is defined by an interaction protocol. There are two important aspects in NGs:

- The agents randomly interact and use a simple set of rules to update their state.
- The agents converge to a consistent state in which all the objects of the set have a uniquely assigned name, by using a distributed social strategy.

Generally, two agents are randomly extracted at each round to perform the role of the "speaker" and the "listener" (or "hearer", as used in [15,1]). The interaction between speaker and listener determines the update of the internal state of the agents. DCSPs and NGs share a variety of common features, as already introduced in [3,4].

2.3 Self-stabilizing Algorithms

The definition of *self-stabilizing algorithm* in distributed computing was first introduced in [7]. A system is *self-stabilizing* whenever each system configuration associated with a *solution* is an absorbing state (global stable state), and any initial state of the system is in the basin of attraction of at least one *solution*.

In a self-stabilizing algorithm, we program the agents of our distributed system to interact with their neighbors. The agents update their state through these interactions by trying to find a stable state in their neighborhood. Since the nature of these algorithms is distributed, many legal configurations of agents states and their neighbors states start arising sparsely. Not all of these configurations are mutually compatible, and, thus, they form mutually inconsistent potential cliques. A self-stabilizing algorithm must find a way to make the global legal state emerge from the competition among these potential cliques. Dijkstra [7] and Collin [6] suggest that an algorithm designed in this way may not always converge, and a special agent is needed to break the system symmetry. [4] shows how a different strategy based on the concept of random behavior and probabilistic transition function can solve specific distributed constraint satisfaction problems with a probability of one. Moreover, [4] shows empirically how this approach can be used on a variety of CSP instances. In Sec. 4.2 we discuss how this later strategy is implemented on Fuzzy CSP instances.

3 Related Work

This paper extends the results in [3,4], in which some of the authors of this paper have solved (crisp) DCSPs with an algorithm inspired by the NG model. Whilst a number of approaches have been proposed to solve DCSPs [14,18] or centralized Fuzzy CSP [14] alone, there is less work in the literature related to solution schemes able to solve CSP instances that are both fuzzy and distributed.

It is important to notice the fundamental difference, with respect to this work, with the DCSP algorithms designed by Yokoo [18]. Yokoo addresses three fundamental kinds of DCSP algorithms: *Asynchronous Backtracking*, *Asynchronous weak-commitment Search* and *Distributed Breakout Algorithm*, all of them also presented in a survey article [18]. Although these algorithms share the property of being asynchronous, they require a pre-agreed agent/variable ordering. The algorithm presented in this paper does not need this initial condition. Therefore, we do not require a pre-processing phase where the ordering is defined, and it also allows for a more dynamic execution, since agents may leave and join without redefining the ordering.

Fuzzy DCSPs has been of interest to the Multi-Agent System community, especially in the context of distributed resource allocation, collaborative scheduling, and negotiation (e.g., [11]). Those works focus on bilateral negotiations and when many agents take part, a central coordinating agent may be required.

For example, the work in [11] promotes a rotating coordinating agent which acts as a central point to evaluate different proposals sent by other agents. Hence, the network model employed in those work is not totally distributed. One more important note is that this work focuses on competitive negotiation, where agents try to outsmart each other (i.e., opposed to our collaborative negotiation).

In [12] the authors propose two approaches to solve these problems: an iterative method and an adaptation of the *Asynchronous Distributed constraint OPTimisation* algorithm (*ADOPT*) for solving Fuzzy DCSP. They also present experiments on the performance comparison between the two approaches, showing that ADOPT is more suitable for low density problems; density is equivalent to the number of links divided by the number of agents.

Finally, in [16,17] the authors define the fuzzy GENET model for solving binary Fuzzy CSPs with a neural network model. Through transforming Fuzzy CSPs into $[0, 1]$ integer programming problems, the authors display the equivalence between the underlying working mechanism of fuzzy GENET and the discrete Lagrangian method. Benchmarking results confirm its feasibility in tackling Fuzzy CSPs, and flexibility in dealing with over-constrained problems. After a number of cycles, the network settles in a stable state. In this stable state, if the obtained fuzzy preference is greater/equal than a predefined threshold α_0, an acceptable solution is considered to be found. Otherwise, the network is trapped in a local minimum. Even if this termination conditions can be implemented in our self-stabilizing algorithm as shown in the tests over the $n \times (n-1)$-queens problem in Sec. 7. In the implementation we propose in Sec. 4 we let the algorithm search for the best optimum of the problem (i.e., without a lower threshold).

4 An Algorithm for Fuzzy Naming Games

In this section we extend classical NGs to take into account fuzzy scores associated with words, therefore, we propose an algorithm that solves Fuzzy NGs. Since we deal with fuzzy values associated only with words, we can consider Fuzzy NGs as particular Fuzzy DCSP instances, $P = \langle X, D, C, A\rangle$ (see Sec. 2.1). In this problem we have fuzzy unary constraints describing the preferences over the possible words, and binary crisp constraints that are satisfied only if the words chosen from two neighboring agents are the same (i.e., $x = y$). In Sec. 5 we further extend the algorithm in order to consider fuzzy binary constraints among agents, and consequently, to solve plain Fuzzy DCSPs.

At each round, the algorithm is based on two kinds of entities. The first is a single *speaker*, which communicates its choice on the word and the related fuzzy preference. The second is a set of *listeners*, which are the speaker's neighboring agents. These neighbors are those agents that can directly communicate with the speaker, through the communication network over the agents. At each round r, an agent is drawn with uniform probability to be the speaker. In the following of this section we describe in detail each step of the interaction scheme that defines the behavior between the speaker and the listeners: we consider three phases, *i) broadcast*, *ii) feedback* and *iii) update*. Each agent marks the element that it expects to be the final shared name in order to recall it when necessary.

4.1 Interaction Protocol

Broadcast. The speaker $a_s \in A$ executes the broadcast protocol. We suppose that each speaker a_s manages a variable $s \in X$. The speaker checks if the marked variable assignment $b \in D$ is in *top*, where *top* is the set of current best assignment, $top = \{x_s | x_s = \arg\max_x[(\bigotimes c_{V_s}\eta[s := x])\Downarrow_s]\}$. The $\bigotimes$ composition is performed over all the constraints that include s in their support V_s, that is $s \in V_s$; then, the result is projected over s (see Sec. 2) in order to obtain a constraint over s only, and, finally, we consider the best preference associated with this constraint (with max). If the current marked variable assignment is not in *top*, the the agent selects a new variable assignment b from *top* with uniform probability, and marks it. The agent recalls the value $u = (\bigotimes c_{V_s}\eta[s := b])\Downarrow_s$. and broadcasts the couple $\langle b, u\rangle$ to all its listeners, that is, it sends its subjective preference for the name of s.

Notice that, even if in this case we only have one unary fuzzy constraint over s, we perform the $\bigotimes$ composition to enforce the consistency w.r.t. not allowed values of s imposed by crisp binary constraints over it. As a remind, crisp binary constraints impose equality among the variables of different agents.

Feedback. All the listeners receive the broadcast message $\langle b, u\rangle$ from the speaker. Each listener $a_l \in A$, which controls variable l, computes the value $(\bigotimes c_{V_l}\eta[s := b][l := d_k])\Downarrow_l$ for all possible d_k values, where d_k is any possible assignment for variable l, and c_{V_l} is any constraint with a scope that includes variable l. In other

words, we compute the combination of the fuzzy preferences (equal to v_k) for each d_k assignment, supposing that the speaker chooses word b. Each listener sends back to a_s a feedback message according to the following two cases:

- *Failure.* If $u > \max_k(v_k)$ there is a *failure*, and the listener feedbacks a failure message containing the maximum value and the corresponding assignment for l, $\mathbf{Fail}\langle\max_k(v_k)\rangle$. This corresponds to a failure because the value proposed by the speaker is better than an upper preference threshold for the same word, computed from the point of view of the listener instead.
- *Success.* If $u \leq \max_k(v_k)$ we are in *success* conditions, the listener feedbacks **Succ** only.

Update. The listener feedback determines the update of a_s and of each a_l that has participated to the interaction. When a listener a_l feedbacks a **Succ**, and if it has an preference value for $d_k = b$ higher than u, then it lowers the preference level for d_k to u. If a_s receives only **Succ** feedback messages from all its listeners, then it does not need to update.

Otherwise, a_s may receive a number $h \geq 1$ of $\mathbf{Fail}\langle v_j\rangle$ feedback messages. In this case, the speaker selects the worst fuzzy preference v_w, s.t. $\forall j, v_w \leq v_j$. As a consequence, a_s sends to all its listeners a $\mathbf{FailUpdate}\langle v_w\rangle$. Thus, the speaker changes the preference for b of its unary constraint $c_{\{s\}}$ with the worst fuzzy level among the failure feedback messages, i.e., $c_{\{s\}}\eta[s := b] = v_w$. In words, it adapts the value of its variable s in accordance to its neighborhood, since fuzzy preferences are composed with the min operator. In addition, each listener a_l sets its preference for word b to v_w, i.e., $c_{\{l\}}\eta[s := b][l := d_l] = v_w$. In words, the feedback of the "worst" listener is propagated to all the listeners of a_s.

4.2 Theorems

In this section we report the lemmas and theorems that lead to the convergence property of the algorithm described in Sec. 4.1: we formally prove that the algorithm always terminates with the best solution, that is the word with the highest fuzzy preference. With Lemma 1 we state that a subset of constraints $C' \subseteq C$ has a higher fuzzy preference w.r.t. C. We say that a fuzzy constraint problem is α-consistent if it can be solved with a level of satisfiability of at least α (see also [2]), that is if a there exists a solution with a preference better than (or equal to) threshold α (with $\alpha \in [0..1]$). Lemma 1 holds because min is a monotonically decreasing function.

Lemma 1 ([2]). *Consider a set of constraints C and any subset C' of C. Then we have $\bigotimes C \leq \bigotimes C'$.*

The speaker selection-rule defines a probability distribution function F that tells us the probability that a certain domain assignment is selected. In Lemma 2 we relate F to the convergence of the algorithm with probability 1, related to the level of satisfiability of the problem.

Lemma 2. *If function F selects only the domain elements with preference level greater then α, then the algorithm converges with probability* 1*, to a solution with a preference greater than α.*

From [3,4] we know that if function F allows a random exploration of the word domain, then the algorithm converges to the same word, but this word may not be the optimal one. If we choose F in order to select only words with a preference greater than α, then the algorithm converges to a solution with a global preference greater than α.

With Prop. 1 and Prop. 2 we prepare the background for the main theorem of this section, that is Th. 1. Proposition 1 describes how the global state of the agents converges, while Prop. 2 states that the algorithm converges with a probability of 1.

Proposition 1. *For round $r \to +\infty$, the weight associated to the optimal solution is equal for all the agents, and it is equal to the minimum preference level of that word.*

Proposition 2. *For any probability distribution F the algorithm converges with a probability of* 1*.*

These two propositions can be derived as proposed in [3,4]. At last, we state that the presented algorithm always converges to the best solution of a Fuzzy DCSP.

Theorem 1. *The algorithm described in Sec. 4.1 always converges to the best solution of the represented Fuzzy NG, i.e., it converges to the solution with the highest fuzzy preference.*

The proof comes from the fact that, *i)* according to Prop. 2, the algorithm always converges, and *ii)* we choose a proper function F as described in Lemma 2.

5 Solving Fuzzy Distributed Constraint Satisfaction Problems as Naming Games

In this section we improve the Fuzzy NG algorithm presented in Sec. 4 in order to solve generic Fuzzy DCSPs instances. To accomplish this, we also consider binary fuzzy constraints instead of crisp ones only, as in Sec. 4. In our algorithm we limit ourselves to unary and binary constraints only because any CSP can be translated to an equivalent one, adopting only unary/binary constraints [14].

As proposed in [18], we assign each variable $x_i \in X$ of $P = \langle X, D, C, A\rangle$ to an agent $a_i \in A$. We assume that each agent knows all the constraints that concern its variables [18]. Each agent $i = 1, 2, \ldots, n$ (where $|A| = n$) searches its own variable domain $d_i \in D$ for an assignment that optimizes P. Each agent has an unary constraint c_i, whose support is defined over its managed variable $x_i \in X$; this unary constraints represent the local agent preference for each variable assignment $d_i \in D$. Each agent can interact only with its neighbors:

we may say that the communication network is determined by the network of binary constraints, since we suppose that an agent $a_i \in A$ can communicate only with an $a_j \in A$ agent sharing a binary constraint with it, i.e., $c_{\{i,j\}} \in C$. Any binary constraint $c_{\{i,j\}}$ returns a preference value in the $[0, 1]$ interval, which states the combined preference over the assignment of x_i and x_j together.

The algorithm is dived into time intervals (we call it a "round"), during which the agents are able to interact and share information on their variable assignments and the mutual constraints. At each round r, an agent is drawn with uniform probability to be the speaker a_s. As in Sec. 4, each speaker has a set of listeners a_l, each of them sharing a binary constraint with a_s. In this algorithm the agents keep a list of speakers' proposals up to the last failed interaction, this list is composed of agent-assignment tuples $S = \{\langle a_{s_1}, b_{s_1}\rangle, \ldots, \langle a_{s_q}, b_{s_q}\rangle\}$. The phases of the algorithm are three as in Sec. 4: *i) broadcast*, *ii) feedback* and *iii) update*.

5.1 Interaction Protocol

Broadcast. The speaker a_s executes the broadcast protocol. The speaker computes $top = \{x_s | x_s = \arg\max_x[(\bigotimes c_{V_s}\eta[s := x]) \Downarrow_s]\}$, as in the previous case 4.1. Then, it checks if the marked variable assignment b is in *top*. If the marked variable assignment is not in *top* it selects a new variable assignment b with uniform probability from *top*, and marks it. Then, the agent recalls the value $u = (\bigotimes c_{V_s}\eta[s := b]) \Downarrow_s$, and a_s sends the couple $\langle b, u\rangle$ to all its listeners. In words, the agent composes all the constraints whose scope contains variable s, that is $s \in V_s$, and it sends its preferred assignment.

Feedback. All the $a_l \in A$ listeners receive the broadcast message $\langle b, u\rangle$ from a_s (with $u = \bigotimes c_{V_s}\eta[s := b]$). Each listener a_l adds $\langle b, u\rangle$ to

$$S = \{\langle a_{s_1}, b_{s_1}\rangle, \ldots, \langle a_{s_q}, b_{s_q}\rangle, \langle a_s, b\rangle\}.$$

Then it computes the value $v_k = (\bigotimes c_{V_l}\eta[s := b_1] \ldots [s := b_q][l := d_k]) \Downarrow_l$ for all the possible d_k values, where d_k is any possible assignment for variable l, and c_{V_l} is any constraint with a scope that includes both the speaker s, and the listener l. Then it computes $d_{\max} = \arg\max_{d_k}(v_k)$ and $v_{\max} = \max_{d_k}(v_k)$. Each listener sends back to a_s a feedback message according to the following two cases:

- *Failure.* If $u > v_{\max}$ we obtain a *failure*, and the listener may only feedback **Fail**$\langle \bigotimes c_{V_l}\eta[s := b][l := d_{\max}]\rangle$.
- *Success.* If $u \leq v_{\max}$, we obtain a *success* for this round, and the listener may feedback **Succ** to the speaker.

Notice that this computation is different from the one in the same phase of the algorithm in Sec. 4.1. In this case, the check has to be computed w.r.t. the composition of all the constraints with variable s in their scope. the reason is that in Fuzzy DCSPs we have fuzzy binary constraints either.

Update. As in Sec. 4.1, the feedback of the listeners determines the update of the listeners and of the speaker itself. When a_l feedbacks **Succ**, and if there is a $\bigotimes c_{V_l}\eta[s := b][l := d_k] > u$, then it sets $\bigotimes c_{V_l}\eta[s := b][l := d_k] = u$. If the speaker receives only **Succ** feedback messages from all its listeners, then it does not need to update and the round ends.

Otherwise, that is if the speaker receives a number $h \geq 1$ of **Fail**$\langle v_j \rangle$ feedback messages. In this case, the speaker selects the worst fuzzy preference v_w, s.t. $\forall j, v_w \leq v_j$. As a consequence, a_s sends to all its listeners a **FailUpdate**$\langle v_w \rangle$.

then the speaker sets $\bigotimes c_s\eta[s := b] = v_w$, as performed in Sec. 4.1. In addition, each listener a_l sets its preference for $s := b$ and $l := d_l$ to v_w, i.e., $c_{\{s,l\}}\eta[s := b][l := d_l] = v_w$. In words, the feedback of the "worst" listener is propagated to all the listeners of a_s. Finally, the speaker and the listeners set $S = \emptyset$.

6 An Example of Algorithm Execution

In this section we show a sample execution of the algorithm for Fuzzy DCSP presented in Sec. 5.1. We consider a problem $P = \langle X, D, C, A \rangle$ with three agents (i.e., $a_1, a_2, a_3 \in A$) and both unary and binary constraints, as defined by the network represented in Fig. 1a. The domain for the variables $x_1, x_2, x_3 \in X$ is $D = \{\triangle, \bigcirc\}$.

When we start executing the algorithm, at round $r = 1$ (whose final state is represented in Fig. 1b) we suppose a_1 is the first agent to be randomly chosen as a speaker. It computes the elements with the highest preference over the constraints c_{V_s}, and fills its list *top* with them (which was previously empty). Since this is the first interaction among the agents, the speaker has no marked element, thus it may only draw an element from *top* with uniform probability. As already introduced, agent a_1 computes $\bigotimes c_{V_{x_1}}\eta$ for all $\forall d \in D$, obtaining that $\bigotimes c_{V_{x_1}}\eta[x_1 := \triangle] = 0.1$, and $\bigotimes c_{V_{x_1}}\eta[x_1 := \bigcirc] = 0.2$. Thus, it marks $\bigcirc$ (marked with an asterisk in Fig. 1b), and choses to broadcast $\langle \bigcirc, 0.2 \rangle$ to its neighbors a_2 and a_3 (the broadcast is underlined in Fig. 1b).

Listener a_2 updates the successful speaker-assignment list $S = \{(a_1, \bigcirc)\}$, then it computes $v_k = \bigotimes c_{V_{x_1,x_2}}\eta[x_1 := \bigcirc][x_2 := d_k]$. For $d_k = \triangle$ it finds $v_1 = \bigotimes c_{V_{x_1,x_2}}\eta[x_1 := \bigcirc][x_2 := \triangle] = 0.3$, and for $d_k = \bigcirc$ it finds $v_2 = \bigotimes c_{V_{x_1,x_2}}\eta[x_1 := \bigcirc][x_2 := \bigcirc] = 0.7$. Thus, a_2 returns **Succ**, since $0.2 \leq max(0.3, 0.7)$.

Simultaneously, listener a_3 updates the successful speaker-assignment list $S = \{(a_1, \bigcirc)\}$, and computes $v_k = \bigotimes c_{V_{x_1,x_3}}\eta[x_1 := \bigcirc][x_3 := d_k]$. For $d_k = \triangle$ it finds $v_1 = \bigotimes c_{V_{x_1,x_3}}\eta[x_1 := \bigcirc][x_3 := \triangle] = 0.5$, and for $d_k = \bigcirc$ it finds $v_2 = \bigotimes c_{V_{x_1,x_3}}\eta[x_1 := \bigcirc][x_3 := \bigcirc] = 0.4$. Thus, it returns **Succ**, since $0.2 \leq max(0.5, 0.4)$.

In the update phase the listeners a_2 and a_3 change the preference levels of all the $v_k > 0.2$ to $v_k = 0.2$, i.e., the value broadcast by a_1 in this round (the changed values are represented in bold in Fig. 1b).

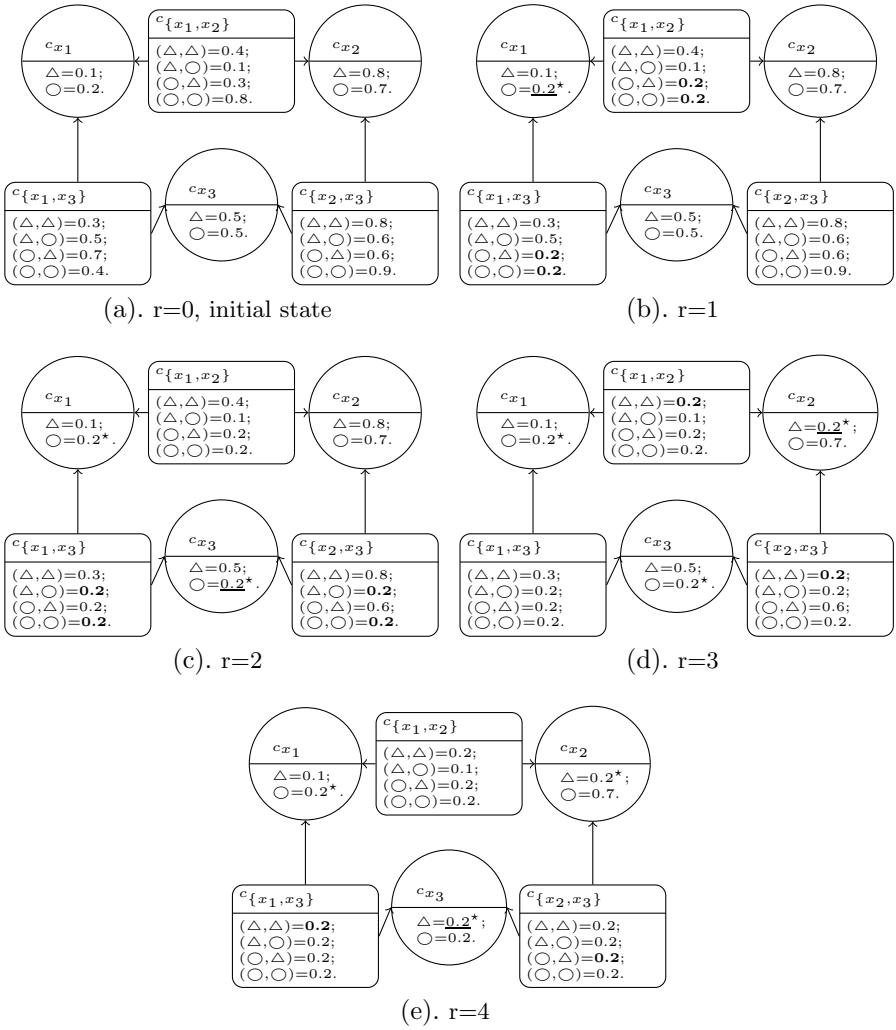


Fig. 1. Example of algorithm run on simple Fuzzy DCSP problem

At $r = 2$ (whose final state is represented in Fig. 1c), agent a_3 is randomly selected. It finds that $\bigotimes c_{V_{x_3}}\eta[x_3 := \triangle] = 0.3$, and $\bigotimes c_{V_{x_3}}\eta[x_3 := \bigcirc] = 0.5$. Thus, it marks $\bigcirc$, and it broadcasts $\langle \bigcirc, 0.5\rangle$ to its listeners a_1 and a_2. Listener a_1 updates the successful speaker-assignment list $S = \{(a_3, \bigcirc)\}$, and computes $v_k = \bigotimes c_{V_{x_1,x_3}}\eta[x_3 := \bigcirc][x_1 := d_k]$. For $d_k = \triangle$ it finds $v_1 = \bigotimes c_{V_{x_1,x_3}}\eta[x_3 := \bigcirc][l := \triangle] = 0.1$, and for $d_k = \bigcirc$ it finds $v_2 = \bigotimes c_{V_{x_1,x_3}}\eta[x_3 := \bigcirc][x_1 := \bigcirc] = 0.2$. Thus, since $0.5 > max(0.1, 0.2)$, a_1 returns **Fail**$\langle \bigcirc, 0.2\rangle$.

Simultaneously, listener a_2 updates the successful speaker-assignment list $S = \{(a_1, \bigcirc), (a_3, \bigcirc)\}$, then computes $v_k = \bigotimes c_{V_{x_1,x_2,x_3}} \eta[x_1 := \bigcirc][x_2 := d_k][x_3 := \bigcirc]$. For $d_k = \triangle$ it finds $v_1 = \bigotimes c_{V_{x_1,x_2,x_3}} \eta[x_1 := \triangle][x_2 := \triangle][x_3 := \bigcirc] = 0.4$, and for $d_k = \bigcirc$ it finds $v_2 = \bigotimes c_{V_{x_1,x_2,x_3}} \eta[x_1 := \bigcirc][x_2 := \bigcirc]\eta[x_3 := \bigcirc] = 0.2$. Thus, since $0.2 \leq max(0.5, 0.2)$, a_2 feedbacks **Succ**.

Since a_3 receives a failure feedback, it calls **FailUpdate(0.2)**. Then, the speaker update its preference level, $\bigcirc = 0.2$. The listeners a_1 and a_2 change their preference levels $v_k = 0.2$ (colored in blue in Fig. 1c). a_1, a_2, and a_3 update their successful speaker-assignment lists $S = \emptyset$.

At round $r = 3$ (whose final state is represented in Fig. 1d), a_2 is the third agent to speak. It finds that $\bigotimes c_{V_{x_2}} \eta[x_2 := \triangle] = 0.4$, and $\bigotimes c_{V_{x_2}} \eta[x_2 := \bigcirc] = 0.2$. Thus, it marks $\triangle$, and it broadcasts $\langle \triangle, 0.4 \rangle$ to agents a_1 and a_3. Listener a_1 updates the successful speaker-assignment list $S = \{(a_2, \triangle)\}$, then computes $v_k = \bigotimes c_{V_{x_1,x_2}} \eta[x_1 := d_k][x_2 := \triangle]$. For $d_k = \triangle$ it finds $v_1 = \bigotimes c_{V_{x_1,x_2}} \eta[x_1 := \triangle][x_2 := \triangle] = 0.1$, and for $d_k = \bigcirc$ it finds $v_2 = \bigotimes c_{V_{x_1,x_2}} \eta[x_1 := \bigcirc][x_2 := \triangle] = 0.2$. Thus, it returns **Fail(b,0.2)**.

Listener a_3 updates the successful speaker-assignment list $S = \{(a_2, \triangle)\}$, then computes $v_k = \bigotimes c_{V_{x_2,x_3}} \eta[x_2 := \triangle][x_3 := d_k]$. For $d_k = \triangle$ it finds $v_1 = \bigotimes c_{V_{x_2,x_3}} \eta[x_2 := \triangle][x_3 := \triangle] = 0.3$, and for $d_k = \bigcirc$ it finds $v_2 = \bigotimes c_{V_{x_2,x_3}} \eta[x_2 := \triangle][x_3 := \bigcirc] = 0.2$. Thus, it returns **Succ**. Since a_2 receives a failure feedback, it calls **FailUpdate(0.2)**. Then, the speaker update its preference level, $\triangle = 0.2$, and the listeners a_1 and a_2 change their preference levels $v_k = 0.2$.

At round $r = 4$ (see Fig. 1e), a_3 is the fourth agent to speak. It finds that $\bigotimes c_{V_{x_3}} \eta[x_3 := \triangle] = 0.3$, and $\bigotimes c_{V_{x_3}} \eta[x_3 := \bigcirc] = 0.2$. Thus, it marks $\triangle$, and it broadcasts $\langle \triangle, 0.3 \rangle$ to a_1 and a_2.

Listener a_1 updates the successful speaker-assignment list $S = \{(a_3, \triangle)\}$, then computes $v_k = \bigotimes c_{V_{x_1,x_3}} \eta[x_1 := d_k][x_3 := \triangle]$. For $d_k = \triangle$ it finds $v_1 = \bigotimes c_{V_{x_1,x_3}} \eta[x_1 := \triangle][x_3 := \triangle] = 0.1$, and for $d_k = \bigcirc$ it finds $v_2 = \bigotimes c_{V_{x_1,x_3}} \eta[x_1 := \bigcirc][x_3 := \triangle] = 0.2$. Thus, it returns **Fail(b,0.2)**. Listener a_2 updates the successful speaker-assignment list $S = \{(a_3, \triangle)\}$, then computes $v_k = \bigotimes c_{V_{x_2,x_3}} \eta[x_3 := \triangle][x_2 := d_k]$. For $d_k = \triangle$ it finds $v_1 = \bigotimes c_{V_{x_2,x_3}} \eta[x_3 := \triangle][x_2 := \triangle] = 0.2$, and for $d_k = \bigcirc$ it finds $v_2 = \bigotimes c_{V_{x_2,x_3}} \eta[x_3 := \triangle][x_2 := \bigcirc] = 0.2$. Thus, it returns **Fail(b,0.2)**. Since a_3 receives two failure feedbacks, it calls **FailUpdate(0.2)**. Then, the speaker updates its preference level, $\triangle = 0.2$, and listeners a_1 and a_2 change their preference levels $v_k = 0.2$.

At round $r = 5$, a_2 is the fifth agent to speak. It finds that $\bigotimes c_{V_{x_2}} \eta[x_2 := \triangle] = 0.2$, and $\bigotimes c_{V_{x_2}} \eta[x_2 := \bigcirc] = 0.2$, thus, $\triangle$ is in *top*. Then a_2 broadcasts $\langle \triangle, 0.2 \rangle$ to a_1 and a_3. Listener a_1 computes $v_k = \bigotimes c_{V_{x_1,x_2}} \eta[x_2 := \triangle][x_3 := d_k]$. For $d_k = \triangle$ it finds $v_1 = \bigotimes c_{V_{x_1,x_2}} \eta[x_2 := \triangle][x_3 := \triangle] = 0.1$, and for $d_k = \bigcirc$ it finds $v_2 = \bigotimes c_{V_{x_1,x_2}} \eta[x_2 := \triangle][x_3 := \bigcirc] = 0.2$. Thus, it returns **Succ**. Listener a_3 computes $v_k = \bigotimes c_{V_{x_2,x_3}} \eta[x_2 := \triangle][x_3 := d_k]$. For $d_k = \triangle$ it finds $v_1 = \bigotimes c_{V_{x_2,x_3}} \eta[x_2 := \triangle][x_3 := \triangle] = 0.2$, and for $d_k = \bigcirc$ it finds $v_2 = \bigotimes c_{V_{x_2,x_3}} \eta[x_2 := \triangle][x_3 := \bigcirc] = 0.2$. Thus, it returns **Succ**. Since all

interactions are successful the speaker calls a success update, the listeners a_1 and a_3 do not change the preference levels, because $v_k \leq 0.2$.

From $r = 6$ the system converges to an absorbing state in which all interactions are success, and the preference levels do not change. This state is also a solution of the fuzzy DCSP.

7 Experimental Results

7.1 Fuzzy NG Benchmarks

In this subsection we show the performance results related to the algorithm presented in Sec. 4. To evaluate different executions we define the probability of a successful interaction at round r, i.e., $P_r(succ)$, given the state of the system in that turn. Notice that with r we mean the current round of speaker/listener interaction: if $r = 2$ it means that we are at the second round. $P_r(succ)$ is determined by the probability that an agent is a speaker s at round r (i.e., $P(s = a_i)$), and by the probability that the agent interaction is successful (i.e., $P_r(succ \mid s = a_i)$). This is computed considering all the n agents participating to the distributed computation:

$$P_r(succ) = \sum_{i=1}^{n} P_r(succ \mid s = a_i) P(s = a_i)$$

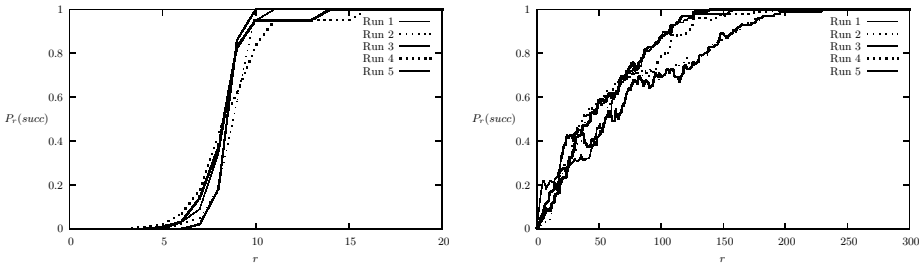
The probability $P_r(succ \mid s = a_i)$ depends on the state of the agent at round r. In particular, it depends on the variable assignment (or word) b selected. Given an algorithm execution, at each round r we can compute $P_r(succ \mid s = a_i)$ over the states of all agents, before that the interaction is performed. Since we have that $P(s = a_i) = 1/n$, we can compute the probability of being in a successful state as:

$$P_r(succ) = \frac{1}{n} \sum_{i=1}^{n} P_r(succ \mid s = a_i)$$

To set up our benchmark, we generate *Random Fuzzy NG* instances (*RFNGs*). To generate such problems, we assign to each agent the same domain of names D, and for each agent and each name in the agent's domain we draw a preference level in the interval $[0, 1]$, by using an uniform distribution. Moreover, RFNGs can only have crisp binary equality-constraints (as defined in Sec. 4). Then, we set the network of agents to be fully connected, in this way, any agent can speak to any agent. We call this kind of problem as *completely connected RFNG instance*, which represents the first set of problems that we use as benchmark. Clearly, by using a completely connected network, the successful global state (where the system is stabilized and a solution is found) is reached very quickly, as it can be seen in Fig. 2a (we discuss this figure in the following).

For the first round of tests, we generate 5 completely connected RFNG instances, with 10 agents and 10 words each (each agent has a word). For each

one of these instances, we compute the best preference level and the word associated to this solution, by using a brute-force algorithm. Then, we execute this algorithm 10 times on each instance. To decide when the algorithm finds the solution, a graph crawler checks marked word of each agent, and the related preference value. If all the agents agree on the marked variable, this means they find an agreement on the name. Then, the graph crawler checks if the shared word has a preference level equal to the best preference (found through the brutal force initial-phase), in such case we conclude that the algorithm has found the optimal solution. In Fig. 2a we can see that the average number of rounds for each of the 5 instances is less than $r = 16$, i.e., within 16 speaker/listeners rounds we can solve all the completely connected RFNG instances.



(a). Completely connected RFNG. (b). Path RFNG.

Fig. 2. Evolution of the average $P_r(succ)$ over 5 different completely connected RFNG instances (2a) and 5 different path RFNG instances (2b). For each instance, we computed the mean $P_r(succ)$ over 10 different runs. We set $n = 10$, and the number of words to 10.

As a second round of tests, we change the topology of our agent networks by defining *Path RFNG* instances [4], which are RFNG instances where the constraint network corresponds to a *path graph*. A path graph (or linear graph) is a particularly simple example of a tree, which has two terminal vertices (vertices that have degree 1), while all others (if any) have degree 2.

In Fig. 2b we report the performance in terms of $P_r(succ)$ for such instances. The instances have been generated following the same guidelines as before: 5 instances with 10 agents and 10 words, and 10 executions for each instance; each preference value in taken from the interval $[0, 1]$, by using an uniform distribution.

As for Fig. 2a, even in Fig. 2b when $P_r(succ) = 1$ the system is in an absorbing state, which we know is also a solution (see Th. 1). As we can notice in Fig. 2b, the network topology among agents strongly influences the performance: having a path graph significantly delays reaching the absorbing state, since we obtain a solution between 140 and 230 speaker/listeners rounds.

In Fig. 3 we show how the *Mean Number of Messages* (*MNM*) needed to find a solution scales over different numbers n of variables in path RFNG instance. For each value n, the *MNM* is measured over 5 different path RFNG instances. We notice that the points approximately overlap the function $cN^{1.8}$.

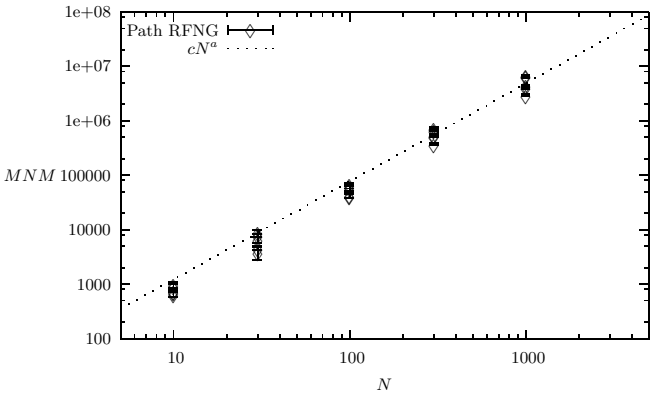


Fig. 3. Scaling of the *MNM* needed to the system to find a solution for different numbers n of variables. For each value n, the *MNM* is measured over 5 different path RFNG instances. We notice that the points approximately overlap the function $cN^{1.8}$.

7.2 Fuzzy CSP Benchmark

For the first fuzzy CSP benchmark, we generate *Random Fuzzy CSP* instances (*RFCSP*). To generate such problems, we consider 10 variables and we assign to each agent a domain of variables D of size 5, and for each assignment we draw a preference level in the interval $[0, 1]$, by using an uniform distribution. Then, we set the binary constraints in such a way that they form a path graph, and we randomly drawn form a uniform distributions in the interval $[0, 1]$ the all the possible fuzzy values of each binary constraint. We call instance a *path RFCSP instance*. In Fig. 4 we show the evolution of the preference level of the solution proposals. In this execution, our algorithm found the best solution after 264 rounds. The level of this solution is 0.4135. We are certain that this is the best solution to the path RFCSP instance because we used a brute-force algorithm to find all the best solutions in advance. It is important to point out that we did not have to set a threshold level, and the algorithm found the best solution autonomously.

For the second fuzzy CSP benchmark we consider the $n \times (n-1)$-queens problem. The $n \times (n-1)$-queens problem [9] is a modification of the n-queens problem in which our objective is to place n queens on a $n \times (n-1)$ chessboard. Because, this board misses a row it is impossible to find a configuration of the queens such that there dos not exist a couple of queens that attack each other. Therefore, we consider a fuzzy version of this problem in which if two queen do not attack each other, their constraint returns a fuzzy preference of one.

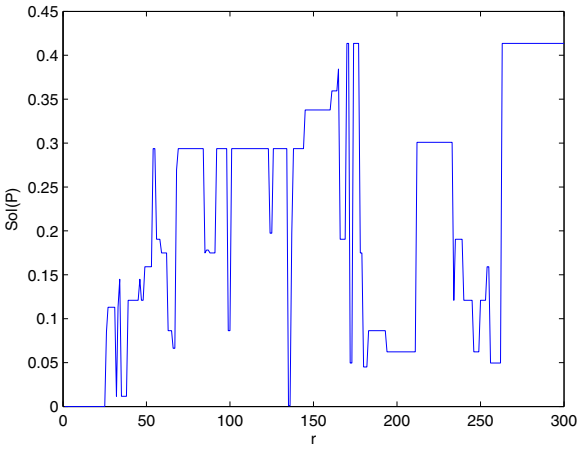


Fig. 4. This graph shows the evaluation of the value of the current solution proposal at each round r for a single algorithm run on a path RFCSP problem, where $n = 10$ and domain size 5

Otherwise, the preference level is proportional to the distance of the queens, according to the formula:

$$\frac{|i_2 - i_1| - 1}{n - 1} \tag{1}$$

Where i_1 is the column of the first queen and i_2 is the column of the second queen. First, as in [17], we search an assignment of the variables in the $n \times (n-1)$-queens problem that has a preference level greater then 0.8.

Table 1. Results on $n \times (n-1)$-queens problem with a threshold of 0.8. The table shows the mean number of rounds *MNR*, and the mean number of messages *MNM* necessary to solve the $n \times (n-1)$-queens problem at different values of n.

n	*MNR*	*MNM*
10	756	20,401
20	1,870	106,590
30	2,130	185,310
40	2,300	269,100
50	2,600	382,200

In Tab. 1, we present the mean number of rounds *MNR*, and the mean number of messages *MNM* necessary to find a solution with a threshold of 0.8 to the $n \times (n-1)$-queens problem. We notice that the algorithm appears to scale well with regards to the increase in the instance size.

Next, Fig. 5 shows the evaluation of the proposed solution at each round r for two algorithms runs on $n \times (n-1)$-queens problem for $n = 8$. Where we did not set a threshold and the algorithm searches for the best possible solution.

We notice that in both executions the algorithm found optimal solutions greater then 0.8. Unfortunately, the algorithm fails to settle on this solution, because we did not set a threshold level and because the algorithm in unable to infer that the optimal solution is lower the 1. To understand that the $n \times (n-1)$-queens problem has global solution smaller then one the algorithm would have compute not only the binary constraint among its variable and its neighbor variable, but also the combination of the binary constraints among the variables of its neighbors. A complete version of this algorithm would consider this constraint or find a way to propagate this constraints as in the Asynchronous Backtracking algorithm [18]. In future work, we intend to implement such complete version. For now it is important to notice that it is unclear if similar algorithms that do not share a variable ordering are able to solve this problem, because in [17] the authors do not consider this situation. Moreover,it is important to point out that in undistributed CSPs it is reasonable to consider various runs at different threshold level to find the best solution. But in a distributed CSP this would require an other level of coordination among the agent that in some circumstances may require costly or unnecessary assumptions on the communication network.

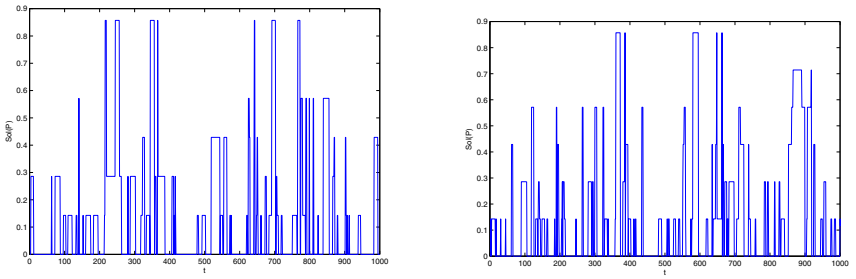


Fig. 5. These two graphs show the evaluation of value of the current solution proposal at each round r for two algorithms runs on $n \times (n-1)$-queens problem with $n = 8$

8 Conclusions and Future Work

In this paper we have shown two main contributions: first we have extended the NG problem [15,1,13,10] to take into account fuzzy preferences over words. Secondly, we have also further extended this algorithm in order to solve a generic instance of Fuzzy DCSPs [14,18,11,12,17], by allowing the solution of binary fuzzy constraints.

Our algorithm is based on the random exploration of the system state-space: our method travels through the possible states until it finds the absorbing state, where it stabilizes. These goals are achieved through the merging of ideas coming from two different fields, and respectively addressed by statistical physics (i.e., NGs), and the computational framework posed by constraint solving (i.e., DCSPs).

The algorithm proposed in Sec. 5 positively answers to an important question: can a distributed uniform probabilistic-algorithm solve general Fuzzy DCSP instances? In other words, we show that a Fuzzy DCSP algorithm may work without a predetermined agent/variable ordering, and it can probabilistically solve instances by taking into account changes to the problem, e.g. deletion/addition of agents during the execution.

Moreover, in the real world, a predetermined agent ordering may be a quite restrictive assumption. For example, we may consider our agents to be corporations, regions in a nation, states in a federation, or independent government agencies. In all of these cases, a predetermined order may not be acceptable for many reasons. Hence, we think it is very important to explore and understand how such distributed systems may work, and what problems may arise.

In the future, we intend to evaluate in depth an asynchronous version of this algorithm, and to test it using comparison metrics, such as a communication cost (number of messages sent) and the *Number of Non-Concurrent Constraint Checks* (*NCCCs*). We would also like to compare our algorithm against other distributed and asynchronous algorithms, such as the *Distributed Stochastic Search Algorithm* (*DSA*) [8], and the *Distributed Breakout Algorithm* (*DBA*) [18]. In addition, we intend to investigate the "fairness" in the loss of privacy between algorithms with no pre-agreed agent/variable ordering, and algorithms with pre-agreed agent/variable ordering. We also plan to develop other functions used to select the speaker in the broadcast phase, and to study the convergence by comparing the performance with the function F used in this paper (see Sec. 4.1).

Finally, we will try to generalise the proposed method to generic semiring-based CSP instances [2], extending the preference from fuzzy to weighted or probabilistic schemes.

References

1. Baronchelli, A., Felici, M., Caglioti, E., Loreto, V., Steels, L.: Sharp transition towards shared vocabularies in multi-agent systems. CoRR, abs/physics/0509075 (2005)
2. Bistarelli, S.: Semirings for Soft Constraint Solving and Programming. LNCS, vol. 2962. Springer, Heidelberg (2004)
3. Bistarelli, S., Gosti, G.: Solving CSPs with Naming Games. In: Oddi, A., Fages, F., Rossi, F. (eds.) CSCLP 2008. LNCS, vol. 5655, pp. 16–32. Springer, Heidelberg (2009)
4. Bistarelli, S., Gosti, G.: Solving distributed CSPs probabilistically. Fundam. Inform. 105(1-2), 57–78 (2010)
5. Bistarelli, S., Gosti, G., Santini, F.: Solving fuzzy DCSPs with naming games. In: IEEE 23rd International Conference on Tools with Artificial Intelligence, ICTAI 2011, pp. 930–931 (2011)
6. Collin, Z., Dechter, R., Katz, S.: On the feasibility of distributed constraint satisfaction. In: IJCAI, pp. 318–324 (1991)
7. Dijkstra, E.W.: Self-stabilizing systems in spite of distributed control. Commun. ACM 17, 643–644 (1974)

8. Fitzpatrick, S., Meertens, L.: An Experimental Assessment of a Stochastic, Anytime, Decentralized, Soft Colourer for Sparse Graphs. In: Steinhöfel, K. (ed.) SAGA 2001. LNCS, vol. 2264, pp. 49–64. Springer, Heidelberg (2001)
9. Guan, Q., Friedrich, G.: Extending Constraint Satisfaction Problem Solving in Structural Design. In: Belli, F., Radermacher, F.J. (eds.) IEA/AIE 1992. LNCS, vol. 604, pp. 341–350. Springer, Heidelberg (1992)
10. Komarova, N.L., Jameson, K.A., Narens, L.: Evolutionary models of color categorization based on discrimination. Journal of Mathematical Psychology 51(6), 359–382 (2007)
11. Luo, X., Jennings, N.R., Shadbolt, N., Leung, H., Lee, J.H.: A fuzzy constraint based model for bilateral, multi-issue negotiations in semi-competitive environments. Artif. Intell. 148, 53–102 (2003)
12. Nguyen, X.T., Kowalczyk, R.: On solving distributed fuzzy constraint satisfaction problems with agents. In: Proceedings of the 2007 IEEE/WIC/ACM International Conference on Intelligent Agent Technology, IAT 2007, pp. 387–390. IEEE Computer Society, Washington, DC (2007)
13. Nowak, M.A., Plotkin, J.B., Krakauer, D.C.: The evolutionary language game. Journal of Theoretical Biology 200(2), 147–162 (1999)
14. Rossi, F., van Beek, P., Walsh, T.: Handbook of Constraint Programming (Foundations of Artificial Intelligence). Elsevier Science Inc., New York (2006)
15. Steels, L.: A self-organizing spatial vocabulary. Artificial Life 2(3), 319–332 (1995)
16. Wong, J., Ng, K., Leung, H.: A Stochastic Approach to Solving Fuzzy Constraint Satisfaction Problems. In: Freuder, E.C. (ed.) CP 1996. LNCS, vol. 1118, pp. 568–569. Springer, Heidelberg (1996)
17. Wong, J.H.Y., Leung, H.: Extending GENET to solve fuzzy constraint satisfaction problems. In: Proceedings of the Fifteenth National/Tenth Conference on Artificial Intelligence/Innovative Applications of Artificial Intelligence, AAAI 1998 IAAI 1998, Menlo Park, CA, USA, pp. 380–385. American Association for Artificial Intelligence (1998)
18. Yokoo, M., Hirayama, K.: Algorithms for distributed constraint satisfaction: A review. Autonomous Agents and Multi-Agent Systems 3, 185–207 (2000)