The impact of wireless communication on distributed constraint satisfaction

Mohamed Wahbi, Kenneth N. Brown
{mohamed.wahbi,ken.brown}@insight-centre.org

Insight Centre for Data Analytics, School of Computer Science and IT,
University College Cork, Ireland

Abstract. Distributed constraint satisfaction (*DisCSP*) models decision problems where physically distributed agents control different decision variables, but must communicate with each other to agree on a global solution. Most DisCSP research assumes an abstract communication layer based on a peer-to-peer wired network. However, many practical applications of distributed reasoning require to be implemented over wireless networks, which impose different communication costs, and may affect the performance of DisCSP algorithms. We study the impact of wireless network topology and routing on two leading DisCSP algorithms – ABT and AFC-ng. We introduce a new framework for experiments which models different communication layers. We show that the communication layer has a significant impact on the messaging costs, which can vary by over an order of magnitude. We also show the impact on computation time, where the equivalent non-concurrent constraint checks can vary by a factor of 6. Finally, we show that given a fixed agent ordering, changing the communications topology can increase the number of messages by up to 50%.

1 Introduction

Distributed Constraint Satisfaction (*DisCSP*) models constrained decision problems where different variables are controlled by physically distributed agents. The agents must communicate with each other to reach a global assignment which satisfies all constraints. The main algorithms and protocols include synchronous [37, 23, 33] and asynchronous [38, 2] search, backtracking, local search [13, 39] and dynamic programming methods [26], and algorithms which respect privacy and autonomy [34, 4, 18] versus those which partially centralise the decision making [20]. The main metrics are non-concurrent constraint checks ($\#ncccs$) [21], which measures the longest sequence of computation among the agents as a proxy for elapsed time, and messages ($\#msg$)[19], which counts the total number of messages exchanged between agents. Communication delays can be modelled by adding extra increments to get $\#encccs$, the equivalent non-concurrent constraint checks [5]. Most research assumes an abstract model of the underlying communication network [41, 30, 17, 32, 11], equivalent to essentially a peer-to-peer wired network. This model works well for applications operating over standard internet architectures.

There is a rich domain of application problems for DisCSP which assume wireless communication, including, for example, dynamic coordination of missions in unmanned aerial vehicles (*UAVs*) [28], mobile robot coordination [6], decision making in

wireless sensor networks (*WSNs*) [1, 15, 36, 3], and dynamic channel selection [25] and time-division scheduling for the self-configuration of ad hoc wireless networks. However, the standard communication layer for DisCSP does not account for all the relevant details of wireless communication, and so algorithms may behave radically differently when implemented on physical devices. For example, in wireless communication, each radio transmission consumes energy, and so the number and size of individual transmissions required to deliver a message is significant, as opposed to the number of end-to-end messages. This is particularly important in remote operation (e.g., UAVs or WSNs) where the agents have limited battery power. In addition, there are many different protocols for exchanging information in a wireless network, ranging from peer-to-peer unicast to one-to-many local broadcast, and from static routing to dynamic routing based on flooding the network. Thus, the communications layer may significantly change the number of individual message transmissions required. More message transmissions means longer delays, and it is known [5, 7] that delays in exchanging messages adversely affects the $\#encccs$ metric in some DisCSP algorithms. To establish DisCSP for wireless applications, we require a better understanding of how the underlying communication layer affects the performance of different algorithms.

In this paper, we propose a new framework for analysing *wireless* DisCSP, based on wireless communication networks. The framework distinguishes between the *constraint graph* and the *communications graph*, which may be different. Direct communication is only possible between adjacent nodes in the communication graph. Exchanging messages between neighbours in the constraint graph thus may require multiple transmissions in the communications graph. We use the framework to re-evaluate the asynchronous ABT algorithm [38, 2] and the partially synchronised AFC-ng [7, 33] algorithm. We consider a range of different communication network topologies, from linear chain trees to complete graphs. We consider the abstract source routing protocols N-way unicast, multicast and multicast* (multicast with local broadcasts), which vary in the number of individual transmissions required to send messages to a set of recipients. We show that changing the topology has a significant impact on the number of message transmissions, sometimes causing an order of magnitude increase for the same algorithm and routing protocol. Similarly, we show that the topology and routing protocol also combine to affect messaging, with N-way unicast on linear topologies requiring significantly more messages than the standard DisCSP model assumes, while multicast* on complete communication networks reduces the number of messages compared to the standard model. Varying the communications layer also has an impact on $\#encccs$, increasing it in the worst case by a factor of six. The performance of static DisCSP ordering heuristics is also influenced by the communication layer, with different topologies increasing the message count by up to 50%. Finally, we propose future directions for wireless DisCSP research, with the eventual aim of modifying the algorithms and ordering heuristics to adapt to different communication layers.

2 Background

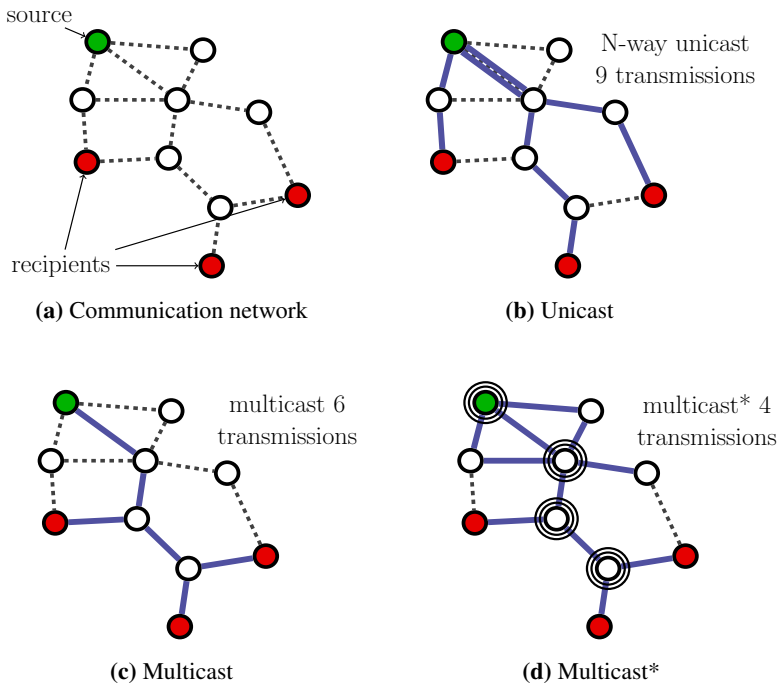
The Distributed Constraint Satisfaction Problem (DisCSP) is a 5-tuple $(\mathcal{A}, \mathcal{X}, \mathcal{D}, \mathcal{C}, \phi)$, where $\mathcal{X}$ is a set of variables $\{x_1, \ldots, x_n\}$, $\mathcal{D} = \{D_1, \ldots, D_n\}$ is a set of domains,

where D_i is a finite set of values from which one value must be assigned to variable x_i, $\mathcal{C}$ is a set of constraints, $\mathcal{A}$ is a set of agents $\{A_1, \ldots, A_a\}$, and $\phi : \mathcal{X} \rightarrow \mathcal{A}$ is a function specifying an agent to control each variable. During a solution process, only the agent which controls a variable can assign it a value. A constraint $C(X) \in \mathcal{C}$, on the ordered subset of variables $X = (x_{j_1}, \ldots, x_{j_k})$, is $C(X) \subseteq D_{j_1} \times \cdots \times D_{j_k}$, and specifies the tuples of values which may be assigned simultaneously to the variables in X. For this paper, we restrict attention to binary constraints. We denote by $C_i \subseteq \mathcal{C}$ all constraints that involve x_i. A *solution* is an assignment to each variable of a value from its domain, satisfying all constraints. Each agent A_i knows all constraints relevant to its variables (C_i) and the other variables involved in its constraints (its *neighbours* in the constraint graph). Without loss of generality, we assume each agent controls exactly one variable ($a = n$), so we use the terms agent and variable interchangeably and do not distinguish between A_i and x_i. A variety of problems have been tackled using DisCSP, including tracking in sensor networks [1], resource allocation [27] and meeting scheduling [22].

Asynchronous Backtracking (ABT) [38, 2] is an asynchronous algorithm executed autonomously by each agent in the problem, and is guaranteed to compute a global consistent solution (or detect inconsistency) in finite time. Each agent proposes values for its own variable to other agents, and reports no-goods. Agents operate asynchronously, but are subject to a known total priority order, o.

Nogood-Based Asynchronous Forward Checking (AFC-ng) [33] is a partially synchronised algorithm that uses no-goods as justification for value removals. Following a total priority agent ordering o, agents assign their variables one by one, recording assignments in a data structure called the Current Partial Assignment (CPA). Once an agent adds its variable assignment to the CPA, it sends the CPA to its unassigned neighbours to perform forward checking (FC [12]) asynchronously. AFC-ng allows different agents to perform backtracks concurrently to the same or different destinations. As a result, several CPAs can be generated simultaneously by the destination agents. Due to the timestamps integrated in the CPAs, the CPA coming from the highest level in the agent ordering will eventually dominate.

In the standard DisCSP communication model [41, 30, 17, 32] each exchange of information from one agent to another (e.g., value choice, acknowledgement, no-good, constraint description) is represented as a *message*. This is an abstraction of the communication layer, based on the assumption that the number of source-to-destination messages is the main factor in communication cost. Zivan and Meisels (2006) proposed an Asynchronous Message Delay Simulator (AMDS) [41] for distributed constraint reasoning algorithms. AMDS is a Mailer thread through which all messages are passed to simulate message delays. The mailer holds a counter of non-concurrent computation steps (LTC, Logical Time Counter) performed by agents, represented as the number of non-concurrent constraint checks ($\#ncccs$). Communication delays can be modelled by adding extra increments to get $\#encccs$. Each agent maintains its own LTC, and attaches it to each message she sends. An agent that receives a message updates her counter to the maximum value between the received LTC and her own counter. Next, she performs a computation step and sends her outgoing messages. Immediately prior sending a message, the agent increments her LTC by the number of constraints checks performed during that step. When an agent desires to send a message she passes it to



(a) Communication network **(b)** Unicast

(c) Multicast **(d)** Multicast*

Fig. 1: Different communication protocols.

the dedicated mailer thread. Upon receiving that message, the mailer updates its LTC by the value of the LTC carried by the message if its value is larger than that held in the mailer. A delay for the message is then chosen and the message is added to the outgoing queue. The queue is ordered by increasing LTC. When a message reaches the front of the queue, it is removed, and delivered to the incoming queue of the receiving agent.

Each agent maintains a count of the messages it sends ($\#msg$), incrementing it by 1 for each message sent to the mailer. Note that each message counts 1 regardless of size, and is thus recording the instance of a single source-to-destination communication. Most experimental comparisons of DisCSP algorithms record the maximum $\#ncccs$ value over all agents at termination, and the sum of the message counts ($\#msg$) over all agents. Occasionally, the total number of constraints checks (summed over all agents) or the number of communication cycles is also reported. The most important metric is usually considered to be $\#ncccs$, consistent with work in general distributed algorithms [19, p. 22]. This model works well with applications implemented over standard wired TCP/IP networks, where variation in routing and packet retransmissions can be averaged over all agents and messages.

There are many practical application problems which rely on wireless networking. For example, in dynamic mission scheduling in unmanned aerial vehicles (*UAVs*), clusters of UAVs are remote from the base and must coordinate their actions and revise their plans through negotiation with each other using wireless communication [28]. In

wireless sensor networks, small sensor nodes must relay sensed data to a base, and may be required to coordinate their sensing in order to ensure the target phenomena is adequately covered [1, 15, 36, 3]. This is further extended to wireless sensor actuator networks, in which some of the nodes can control aspects of the physical environment. In addition, the operation and configuration of ad hoc and mesh wireless networks requires individual radio nodes to sense the topology of the network and to agree spectrum use, time schedules for communication, and routing paths [25], all of which involve distributed combinatorial problems.

In wireless communication [24], each individual transmission of data consumes energy in order to radiate and receive the signal. The energy consumed depends on the distance over which the data is transmitted, and on the amount of data. In wireless sensor networks, the cost of transmitting one bit of information is estimated to be equal to the cost of executing 1000 to 2000 logical instructions on the sensor node [14, p. 104], and so computation is considered much cheaper than communication. For battery powered nodes, limiting communication cost is the key to maintaining node, and thus network, lifetime. In multi-hop networks (e.g., wireless sensor networks or ad hoc networks), some nodes are not in range of each other, and so intermediate nodes must receive and re-transmit the data in order for it to be delivered. Thus, the delivery of a single message between nodes A_i and A_j may require significantly more transmissions than the same message delivered from A_i to A_k. Also, wireless communication is subject to radio interference, and so messages may have to be retransmitted in order to be received successfully; alternatively, to avoid interference, nodes may have to wait until the radio spectrum is free before they transmit. Thus, as well as incurring additional energy costs, longer transmission paths impose additional delays on message delivery.

Within a multi-hop network, there are many different approaches for routing the data [16]. Methods include flooding the network with messages, decentralised routing with each node maintaining a routing table, and source-level routing, in which each node decides upon the end-to-end route it will use for each recipient. Within source-level routing, options include *N-way unicast*, *multicast*, and multicast using the broadcast medium (which we refer to as *multicast**)(Fig. 1). For N-way unicasting, a source sending to N recipients creates N copies of the message, and then initiates each message along its chosen route. In multicasting, the source constructs a rooted tree with itself as root and containing all the intended recipients. Messages are then sent down the tree, with multiple copies only created when multiple branches leave from a single node. Multicast* takes advantage of the fact that multiple nodes can receive a single transmission, and thus each node in the multicast tree only needs to transmit a single copy of the message (assuming an omnidirectional antenna).

The differences between the standard DisCSP communication model and the wireless communication model raises the question of how our algorithms perform when deployed on wireless networks. The standard DisCSP model is essentially N-way unicasting, either on a complete communication network or on the constraint graph. Every edge in the graph requires the same energy for a transmission – any agent can communicate directly with any neighbour, for a cost of 1 message (plus a delay increment of δ) for each communication. In a problem instance with n agents, sending a variable assignment to m (constraint graph) neighbours takes m transmissions each with delay δ. Even

if we assume that each transmission does consume the same energy, implementing the same instance on a wireless network may incur different costs. If the communication topology is a linear chain, with the source at one end and the recipients at the other, the same variable assignment would require $n*m-(m(m-1)/2)$ transmissions, with the longest delay $n*\delta$. Multicast on the same topology would require just n transmissions, but with the longest delay still $n*\delta$. Finally, if the topology is a complete graph, using multicast*, then we require just 1 transmission, with delay δ. Since increasing delays in messages are known to adversely affect $\#encccs$ for DisCSP algorithms, we may also see variation in the $\#encccs$ metric as we vary the communications layer. Therefore, if we are to deploy DisCSP algorithms on wireless networks, we need to revisit the algorithms, and assess their performance under different communication assumptions.

3 Network Communication Simulator Framework (NeCoS)

The standard DisCSP model views agents as distributed autonomous entities. Almost all distributed constraint reasoning simulators implement agents as Java Threads [40, 30, 17, 32, 11]. Zivan and Meisels (2006) proposed AMDS counting non-concurrent constraints-checks ($\#ncccs$) for systems with message delays. In AMDS, agents run concurrently, exchanging messages using a common mailer. In this section we generalize AMDS to simulate different communication topologies and routing protocols (unicast, multicast, and multicast*). We call the new simulator Network Communication Simulator (*NeCoS*). NeCoS is a Thread to which all messages are passed to simulate delays in communication networks using different communication protocols. We assume two graphs: (i) the standard DisCSP constraint graph, where two agents are neighbours if and only if they share a constraint, and (ii) the communications graph, where two nodes are neighbours if and only if they can communicate directly with each other in a single transmission. There is a function from the constraint graph vertex set (agents) to the communications graph vertex set (nodes), but the edge sets may be arbitrarily different. NeCoS requires as input the communications graph and the function. When A_i sends a message to A_j, the message must traverse a path in the communications graph, which may require multiple retransmissions of the message.

As in AMDS, NeCoS maintains a Logical Time Counter (LTC), which measures the longest sequence of computation and communication between agents. Each agent maintains its own counter. To simulate delays on message transmissions, each message in the system carries the LTC value of its sender. Whenever an agent receives a message, it updates its counter to the maximum value of the received LTC and its own counter. It then performs its computation step and sends messages with the value of its counter incremented by the amount of computation required during this step.

The NeCoS pseudo-code is presented in Figs. 2 and 3. In initialisation, NeCoS stores the communications graph in $network$, and then computes all shortest paths using [8]. When an agent desires to send a message msg to a set of recipients, it emits the message to NeCoS by calling `sendMessage`$(msg, recipients)$. Depending on the routing protocol used, NeCoS runs different procedures (lines 15-17):

unicast: for each recipient, NeCoS creates a copy (m) of the original message and computes the routing tree for that copy. The routing tree is the shortest path in

```
procedure NeCoS()
01.  outQueue ← ∅; LTC ← 0; end ← false ;
02.  network ← getCommunicationGraph() ;
03.  network.computeAllShortestPaths();          /* Use Floyd-Warshall [8, 35] */
04.  while ( ¬end ) do
05.    if ( all agents are terminated ) then end ← true ;
06.    else if ( all agents are idle ) then LTC ← outQueue.first().getLTC();
07.    deliverMessages() ;
procedure deliverMessages()
08.  foreach ( msg ∈ outQueue s.t. msg.getLTC() < LTC ) do
09.    tree ← msg.gettree() ;
10.    A_s ← tree.getRoot() ;
11.    if ( A_s ∈ msg.getRecipients() ) then deliver(msg) to A_s;
12.    if ( routingProtocol ≠ multicast* ) then routingMessage(msg, tree, A_s) ;
13.    else routingMulticast*(msg, tree, A_s) ;
procedure sendMessage(msg, recipients)
14.  switch ( routingProtocol ) do // routingProtocol ∈{unicast,multicast,multicast* }
15.    unicast      : sendUnicast(msg, recipients) ;
16.    multicast    : sendMulticast(msg, recipients) ;
17.    multicast*   : sendMulticast*(msg, recipients) ;
procedure sendUnicast(msg, recipients)
18.  A_s ← msg.getSender();
19.  foreach ( A_i ∈ recipients ) do
20.    A_s.nbMsgSent ← A_s.nbMsgSent + 1;
21.    m ← msg;
22.    m.setRecipients(A_i);
23.    tree ← network.shortestPath(A_s, A_i) ;
24.    m.setRoutingTree(tree);
25.    chooseDelay(m) ;
26.    outQueue.add(m) ;
procedure sendMulticast(msg, recipients)
27.  A_s ← msg.getSender() ;
28.  tree ← network.steinerTree(A_s, recipients) ;
29.  routingMessage(msg, tree, A_s) ;
procedure sendMulticast*(msg, recipients)
30.  A_s ← msg.getSender() ;
31.  tree ← network.steinerTree(A_s, recipients) ;
32.  routingMulticast*(msg, tree, A_s) ;
procedure routingMessage(msg, tree, node)
33.  foreach ( subtree ∈ tree.getSubtreesOf(node) ) do
34.    node.nbMsgSent ← node.nbMsgSent + 1 ;
35.    m ← msg;
36.    m.setRecipients(recipients ∩ subtree);
37.    m.setRoutingTree(subtree);
38.    chooseDelay(m) ;
39.    outQueue.add(m) ;
```

Fig. 2: Network Communication Simulator (Part 1).

```
procedure routingMulticast*(msg, tree, node)
40.  node.nbMsgSent ← node.nbMsgSent + 1 ;
41.  chooseDelay(msg) ;
42.  foreach ( subtree ∈ tree.getSubtreesOf(node) ) do
43.    m ← msg ;
44.    m.setRecipients(recipients ∩ subtree);
45.    m.setRoutingTree(subtree);
46.    outQueue.add(m) ;
procedure chooseDelay(msg)
47.  LTC ← max(LTC, msg.getLTC()) ;
48.  msg.LTC ← msg.getLTC() + δ ;
```

Fig. 3: Network Communication Simulator (Part 2).

the communications graph from the sender to the recipient (line 23). Then, NeCoS calls procedure `chooseDelay`(m) (line 25) to select a random delay needed to transmit the message to the next node in the routing tree (lines 47-48). The copy of the message is then added to the outgoing message queue (i.e., $outQueue$, line 26).

multicast: NeCoS constructs a rooted Steiner tree with the source (A_s) as root and containing all recipients, line 28.[1] Then, NeCoS mimics the multicast routing protocol (`routingMessage` call, line 29). In `routingMessage`, a message is transmitted from the root $node$ to each of its children (roots of its sub-trees, $subtree$, line 33) in the routing tree, $tree$, and each of these in turn queues the message for retransmission to its children (lines 38-39). We increment the number of messages transmitted by $node$ by the number of its children in routing tree (line 34).

multicast*: NeCoS constructs a rooted Steiner tree with the source (A_s) as root and containing all recipients, line 31. Then, NeCoS mimics the multicast* routing protocol (`routingMulticast*` call, line 32). In `routingMessage`, a message is transmitted from the root $node$ to each of its children (roots of its sub-trees, $subtree$, line 42) in the routing tree, $tree$ requiring only one transmission (line 40) from $node$ with the same delay (line 41). However, to simulate this, NeCoS creates a copy m of msg for all children of $node$ in the routing tree, $tree$ (lines 43) and each of these in turn will queue the copy for retransmission to its children (line 46).

In the three communication protocols, the LTC of each transmitted message is updated to the sum of the value of the message LTC and a random selected delay (line 48). Then, the message is added to the outgoing queue ($outQueue$). The outgoing queue is a priority queue in which messages are sorted by their LTC, so that the first message is the message with the lowest LTC.

When there are no incoming messages, and all agents are idle, NeCoS increases the value of its LTC to the LTC value of the first message in the outgoing queue (line 6) and calls procedure `deliverMessages` (line 7). When `deliverMessages` is invoked by NeCoS all messages carrying an LTC smaller than the counter of the simulator are transmitted line 4. Thus, this message is delivered to the root of the routing tree if it is

[1] In our experiments, we use a heuristic algorithm to compute good Steiner trees.

one of the final recipients of that message, line 11. Next, we simulate a routing node for that agent depending on the routing protocol used (lines 12-13).

4 Experiments

In this section we evaluate ABT and AFC-ng under different network conditions. Algorithms are tested on the same static agent ordering using the *max-degree* heuristic. For ABT we implemented an improved version of Silaghi's solution detection [29] and counters for tagging assignments. All experiments were performed on the DisChoco 2.0 platform [32],[2] in which agents are simulated by Java threads that communicate only through message passing.

The algorithms are tested on uniform binary random DisCSPs which are characterized by $\langle n, d, p_1, p_2 \rangle$, where n is the number of agents/variables, d the number of values per variable, p_1 is the constraint graph connectivity defined as the ratio of existing binary constraints to possible binary constraints, and p_2 is the constraint tightness defined as the ratio of forbidden value pairs to all possible pairs. We solved instances of two classes of constraint graphs: sparse constraint graphs $\langle 20, 5, 0.2, p_2 \rangle$ and dense ones $\langle 20, 5, 0.7, p_2 \rangle$. We varied the tightness from 0.1 to 0.9 by steps of 0.1. For each pair of fixed density and tightness (p_1, p_2) we report the average over 20 instances.

We evaluate the performance of the algorithms by communication load and computation effort. Communication load is measured by the total number of transmission messages in the communication network during algorithm execution ($\#transmission$). Computation effort is measured by the average of the equivalent non-concurrent constraint checks ($\#encccs$) [5] that extends the non-concurrent constraint checks ($\#ncccs$) [9]. The $\#encccs$ are a weighted sum of processing and communication time. We simulate uniform random message delays on the communication network. For each message a delay is randomly chosen between 10 and 100 constraint checks. We simulated three communication protocols: unicast, multicast, and multicast*.

To assess the behaviour of ABT and AFC-ng on different communication layers we generate 8 different connected network topologies, using only agents in the problem:
complete: the communication network is a complete graph, where all agents are connected to each other – the maximum number of hops between any two agents is one;
constraint: the communication network matches the constraint graph exactly;
star: the communication network has a star topology where one randomly selected agent is directly connected to all other agents, and there are no other connections;
random (0.7): dense random communication networks, where exactly $0.35 * (n(n-1))$ randomly selected binary connections are created;
random (0.2): sparse random communication networks, where exactly $0.1 * (n(n-1))$ randomly selected binary connections are created;
spanning: the communication network is a random spanning tree that spans the agents of the problem;
ring: the communication network is a random ring, and so the maximum distance between any agent pair is $n/2$ hops;

[2] http://dischoco.sourceforge.net/

Table 1: Performance on hard region when simulating unicast communication protocol.

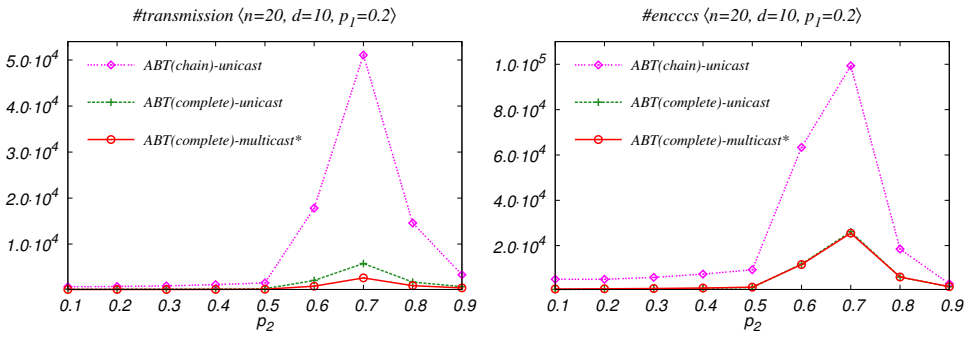
Communication graph	$(p_1 = 0.2, p_2 = 0.7)$				$(p_1 = 0.7, p_2 = 0.3)$			
	$\#transmission$		$\#encccs$		$\#transmission$		$\#encccs$	
Algorithm	ABT	AFC-ng	ABT	AFC-ng	ABT	AFC-ng	ABT	AFC-ng
complete	5 769	**2 724**	26 143	**25 968**	627 742	**188 934**	1 784 567	**1 223 455**
constraint	7 834	**2 886**	**27 807**	30 115	772 906	**182 744**	1 836 074	**1 292 729**
star	12 061	**5 076**	**39 587**	48 301	1 283 398	**361 580**	2 286 323	**2 181 065**
random (0.7)	7 846	**3 518**	**29 640**	32 344	846 561	**242 956**	1 929 618	**1 505 303**
random (0.2)	15 299	**6 147**	**43 790**	55 555	1 609 192	**431 490**	**2 476 220**	2 541 963
spanning	28 236	**10 936**	**67 390**	97 036	3 083 883	**739 513**	**3 673 841**	4 158 617
ring	40 274	**13 654**	**86 113**	120 862	4 602 738	**972 699**	**4 564 197**	5 528 475
chain	51 075	**19 872**	**99 331**	164 056	6 183 110	**1 321 164**	**5 262 304**	6 680 264

Table 2: Performance on hard region when simulating multicast communication protocol.

Communication graph	$(p_1 = 0.2, p_2 = 0.7)$				$(p_1 = 0.7, p_2 = 0.3)$			
	$\#transmission$		$\#encccs$		$\#transmission$		$\#encccs$	
Algorithm	ABT	AFC-ng	ABT	AFC-ng	ABT	AFC-ng	ABT	AFC-ng
complete	5 604	**2 774**	**26 187**	26 758	624 311	**188 967**	1 776 765	**1 241 699**
constraint	6 629	**2 764**	**28 361**	29 793	678 895	**183 789**	1 832 203	**1 279 464**
star	8 657	**3 483**	**38 680**	46 047	881 339	**215 746**	2 276 400	**2 051 918**
random (0.7)	6 976	**3 103**	**29 708**	32 157	736 382	**197 971**	1 929 399	**1 527 361**
random (0.2)	12 022	**4 442**	**45 536**	55 573	1 153 096	**242 460**	2 625 804	**2 588 631**
spanning	19 066	**6 497**	**66 879**	91 341	1 741 666	**304 504**	**3 610 608**	3 823 730
ring	28 641	**8 086**	**90 159**	121 029	2 593 760	**367 807**	**4 843 591**	5 465 870
chain	32 553	**10 917**	**97 718**	153 332	3 055 454	**430 339**	**5 182 070**	5 984 795

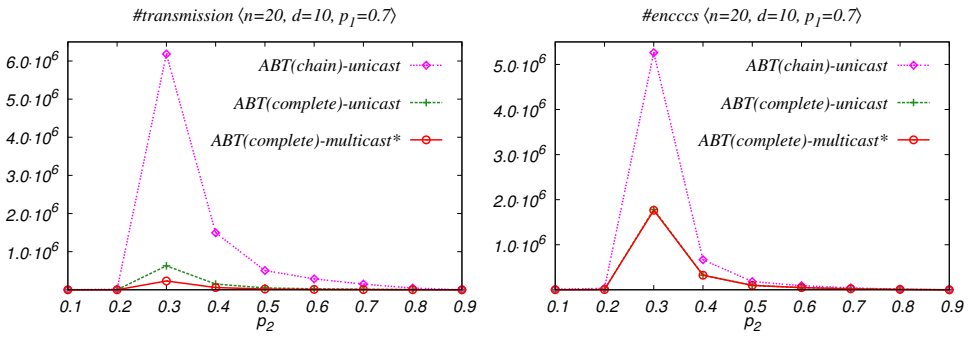
Table 3: Performance on hard region when simulating multicast* communication protocol.

Communication graph	$(p_1 = 0.2, p_2 = 0.7)$				$(p_1 = 0.7, p_2 = 0.3)$			
	$\#transmission$		$\#encccs$		$\#transmission$		$\#encccs$	
Algorithm	ABT	AFC-ng	ABT	AFC-ng	ABT	AFC-ng	ABT	AFC-ng
complete	2 643	**993**	**25 493**	25 959	230 833	**35 028**	1 764 498	**1 113 742**
constraint	4 167	**1 273**	**27 977**	30 186	313 675	**37 869**	1 817 239	**1 181 323**
star	5 416	**1 805**	**37 564**	44 317	487 496	**67 289**	2 242 006	**1 905 299**
random (0.7)	4 013	**1 554**	**29 361**	32 211	366 740	**62 809**	1 913 308	**1 407 144**
random (0.2)	9 004	**3 257**	**44 894**	56 056	874 364	**151 131**	2 613 529	**2 505 835**
spanning	15 522	**5 382**	**65 817**	91 692	1 442 528	**213 293**	**3 600 303**	3 728 196
ring	25 648	**7 741**	**88 114**	122 376	2 517 252	**351 788**	**4 826 414**	5 452 225
chain	31 882	**10 580**	**97 689**	153 234	2 972 596	**415 085**	**5 161 530**	5 977 598



(a) ABT on sparse problems ($\#transmission$)

(b) ABT on sparse problems ($\#encccs$)



(c) ABT on dense problems ($\#transmission$)

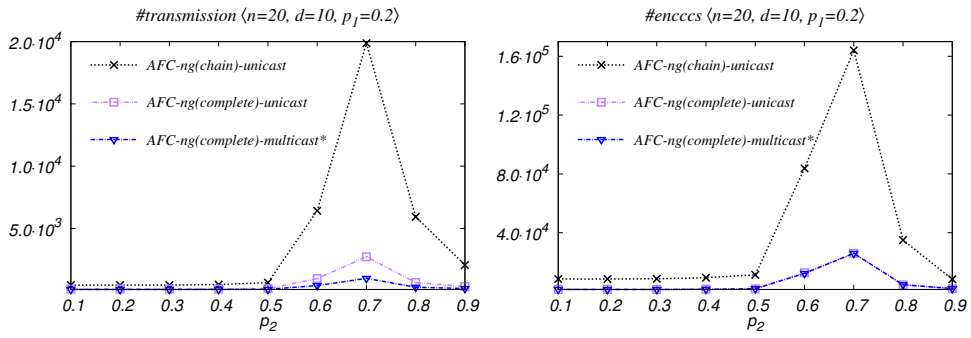
(d) ABT on dense problems ($\#encccs$)

Fig. 4: Comparison of the performance of ABT using unicast on chain communication tree and complete graph, and using multicast* on complete graph.

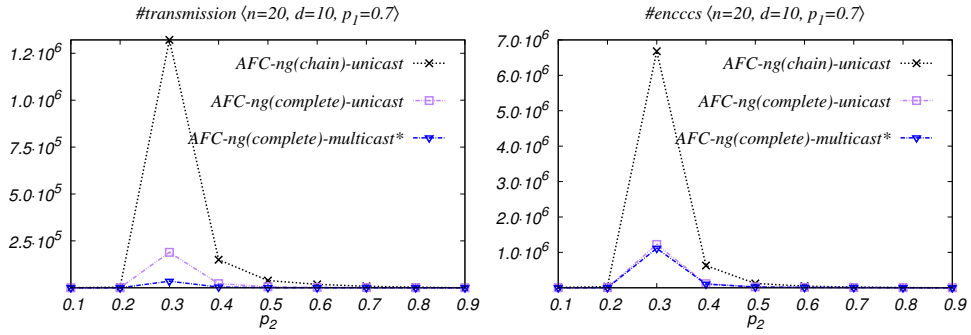
chain: the communication network is a linear chain tree, randomly selected; agents at the ends of the chain are $(n-1)$ hops from each other.

For each topology (except complete and constraint), we generated 5 different random communication networks, and solve each problem instance on each one. The results presented are averaged over 20 problem instances.

For space reason, we show only a subset of the obtained results. Tables 1, 2, 3 present the obtained results on the hard regions $(p_1 = 0.2, p_2 = 0.7)$ and $(p_1 = 0.7, p_2 = 0.3)$. On dense problems on complete communications graphs with multicast* (Table 3), ABT can use 7 (resp. 1.7) times more $\#transmission$ (resp. $\#encccs$) than AFC-ng. On sparse problems on complete graphs when simulating multicast*, ABT requires 2.5 times more $\#transmission$ than AFC-ng while the $\#encccs$ value for both algorithms is similar. On dense problems on chain communications network with multicast, (Table 2), ABT requires 7 times more $\#transmission$ than AFC-ng, but ABT requires slightly fewer $\#encccs$. On sparse problems on chain communications network with multicast, ABT requires 3 times more $\#transmission$ than AFC-ng, but AFC-ng records 1.5 times more $\#encccs$. For unicast (Table 1) we see a similar pattern, but smaller differences. For sparse problems on the chain communication



(a) AFC-ng on sparse problems ($\#transmission$) **(b)** AFC-ng on sparse problems ($\#encccs$)



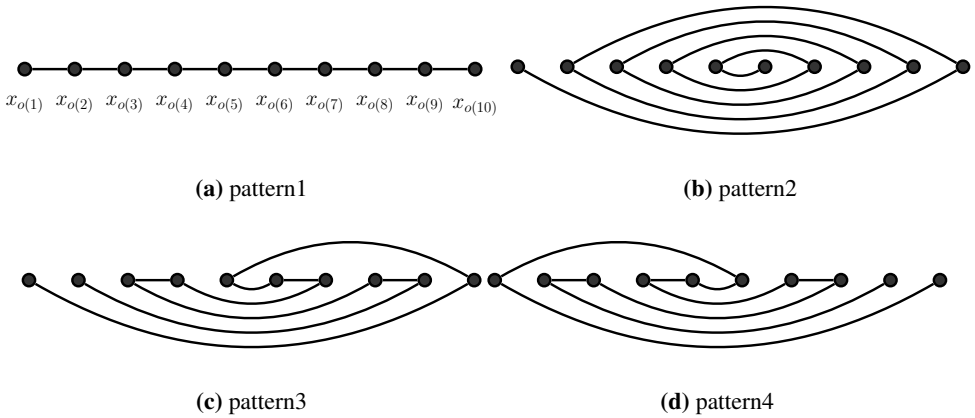
(c) AFC-ng on dense problems ($\#transmission$) **(d)** AFC-ng on dense problems ($\#encccs$)

Fig. 5: Comparison of the performance of AFC-ng using unicast on chain communication tree and complete graph, and using multicast* on complete graph.

networks ABT requires 2.5 times more $\#transmission$ than AFC-ng, but AFC-ng performs 1.6 times more $\#encccs$. Thus, when we use unicasting on sparse communications networks, ABT is better on $\#encccs$, otherwise, AFC-ng is better, and particularly for dense communication networks with multicast*, where ABT requires 7 times more $\#transmission$ than AFC-ng.

We note that, for the two algorithms we studied, changing just the routing protocol rarely changes the ranking, but it does affect the margin of improvement. For example, for multicast on dense networks that match the constraint graph, the ratio of $\#transmission$ for ABT against AFC-ng is 3.7, but for multicast* on the same problems, the ratio increases to 8.2. A change in the topology of the communications graph does affect the ranking for $\#encccs$. For example, for unicast on dense problems, on complete networks AFC-ng offers an improvement over ABT of 45%, but for chain tree networks, ABT is better by a factor of 27%.

Looking at just ABT, for sparse problems, for different communication layers, the $\#transmission$ can vary by a factor of 20 (multicast* on complete, vs unicast on chain), Fig. 4. If we assume unicast on a complete network is the standard DisCSP model, then varying the communication layer could drop $\#transmission$ by a factor



(a) pattern1 **(b)** pattern2

(c) pattern3 **(d)** pattern4

Fig. 6: Patterns used to generate communication chain trees from the max-degree ordering o.

of 2.5, or raise it by a factor of 8. For dense problems, the biggest factor is 25 for $\#transmission$ (this factor can drop by 1/3 or rise by 9), and 2.7 for $\#encccs$.

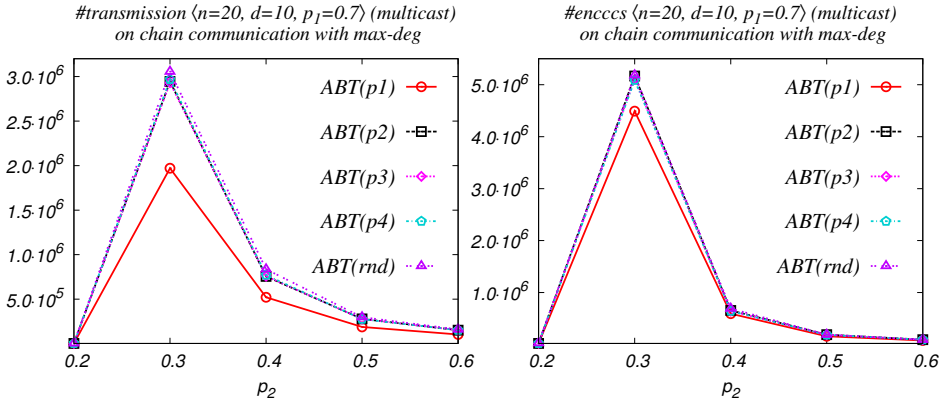
Looking at just AFC-ng on sparse problems (Fig. 5), again the range in $\#transmission$ varies by a factor of 20, and for $\#encccs$ by a factor of 6.5. For dense problems, the variation factor for $\#transmission$ is 37 when comparing the multicast* protocol on complete communications graph to the unicast protocol on chain communications network. The variation is 6 for $\#encccs$. These results show that the $\#transmission$ for AFC-ng using the standard model used so far to compare DisCSP algorithms (unicast on complete communications graph) could be factor of 6 too high or a factor of 6 too low.

Changing just the routing protocol has only a small effect for ABT, varying by a factor of approximately 2, but a larger effect for AFC-ng, of up to 5.5. We believe this is because of the nature of the algorithms. Multicast* does not change the number of no-goods, and ABT sends significantly more no-goods than AFC-ng. In general, multicasting (multicast and multicast*) offers an improvement over unicast when the communications graph is sparse, while multicast* improves over multicast when the communications graph is dense.

On the chain communications network, AFC-ng appears to require more $\#encccs$ than ABT, while ABT requires more $\#transmission$. This is investigated more closely in the next section. On chain communications, multicast and multicast* are similar, apart from random variation in message delays.

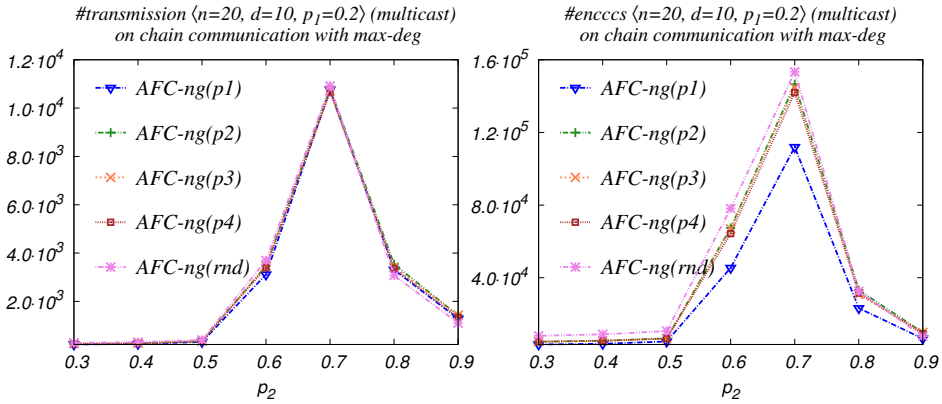
4.1 Communication chain trees

In the following we evaluate the performance of ABT and AFC-ng on different chain communication networks. Based on the max-degree ordering (o) (computed from the constraint graph) 5 patterns are used to generate chain communication trees (T). In the following we denote by $x_{o(k)}$ the k^{th} agent in o. This patterns are presented on Fig. 6.



(a) $\#transmission$ (multicast) **(b)** $\#encccs$ (multicast)

Fig. 7: Performance of ABT on dense problems, multicast routing protocol, different communication chain trees.



(a) $\#transmission$ **(b)** $\#encccs$

Fig. 8: Performance of AFC-ng on sparse problems, multicast routing protocol different communication chain trees.

p1: every pair of adjacent agents in o are connected in the chain communication tree ($T_E = \{\{x_{o(k)}, x_{o(k+1)}\} \mid k \in 1..n-1\}$.

p2: for each pair of adjacent agents on the resulting communication tree T, we try to maximise their separation in the ordering o. Thus, the first agent ($x_{o(1)}$) is connected to the last ($x_{o(n)}$), the last the second, and so on: $T_E = \{\{x_{o(1)}, x_{o(n)}\}; \{x_{o(n)}, x_{o(2)}\}; \{x_{o(2)}, x_{o(n-1)}\}; \ldots\}$.

p3: we try to maximise the distance between the first and the second agent on o. Thus, $x_{o(1)}$ and $x_{o(2)}$ are the extremities of the resulting chain communica-

tion tree T, $T_E = \{\{x_{o(2)}, x_{o(n-1)}\}; \{x_{o(n-1)}, x_{o(n-2)}\}; \{x_{o(n-2)}, x_{o(3)}\}; \ldots; \{x_{o(\frac{n}{2})}, x_{o(n)}\}; \{x_{o(n)}, x_{o(1)}\}\}$.

p4: we try to maximise the distance between the last and the second last agent on o. Thus, $x_{o(n-1)}$ and $x_{o(n)}$ are the extremities of the resulting chain communication tree T, $T_E = \{\{x_{o(n-1)}, x_{o(2)}\}; \{x_{o(2)}, x_{o(3)}\}; \{x_{o(3)}, x_{o(n-2)}\}; \ldots; \{x_{o(\frac{n}{2}+1)}, x_{o(1)}\}; \{x_{o(1)}, x_{o(n)}\}\}$.

rnd: we generate random chain trees.

Again we show only a subset of results due to space reasons. Looking at ABT performance on dense problems (Fig. 7), when the chain communication tree does not match the max-degree agent ordering (pattern 1) the $\#transmission$ required is increased by 50%. For $\#encccs$ a small improvement can be seen for pattern 1 compared to other patterns. For AFC-ng on sparse problems (Fig. 8), the pattern used doesn't really matter for the $\#transmission$. For $\#encccs$, if the chain communication fits the max-degree we get an improvement of 30%.

5 Conclusion

There are many potential applications of Distributed Constraint Satisfaction that rely on wireless networking for the agent to communicate. The standard DisCSP communication model does not represent important features of wireless communication, particularly the topology of the communications graph and the routing protocols. We have introduced a new simulator for modelling wireless communication in DisCSP, NeCoS, which allows different topologies and routing protocols to be modelled. We have shown that varying the communications layer can have a significant effect on the performance metrics for existing DisCSP algorithms, sometimes varying the number of messages by a factor of over 30. The topology of the network also has an impact on the performance of the algorithms, causing a variation of up to 50% in the number of transmissions for ABT and almost 30% in the number of $\#encccs$ for AFC-ng.

These results indicate that, if DisCSP is to be applied to wireless network applications, further research is required on the interaction between the algorithms and the communications layer. In particular, we will investigate ordering heuristics that adapt to the wireless network structure. We will explore algorithm variants that exploit the communication structure; for example, in distributed optimisation, we believe the broadcast mechanism of the AFB family [10, 31] will work well with multicast*. Finally, we will extend these ideas to explore more features of wireless networking, including cases where the communication network is dynamic or agents are mobile, and so areas of the network may become temporarily disconnected [28].

Acknowledgements This work is funded by Science Foundation Ireland (SFI) under Grant Number SFI/12/RC/2289. We are grateful to Cormac Sreenan for many conversations about wireless networking, and to Pedro Meseguer and the anonymous reviewers for their helpful comments on this submission.

Bibliography

[1] Béjar, R., Domshlak, C., Fernández, C., Gomes, C., Krishnamachari, B., Selman, B., Valls, M.: Sensor networks and distributed csp: communication, computation and complexity. Artif. Intel. 161, 117–147 (2005)

[2] Bessiere, C., Maestre, A., Brito, I., Meseguer, P.: Asynchronous backtracking without adding links: a new member in the ABT family. Artif. Intel. 161, 7–24 (2005)

[3] Bijarbooneh, F.H., Flener, P., Ngai, E., Pearson, J.: Optimising quality of information in data collection for mobile sensor networks. In: Quality of Service (IWQoS), 2013 IEEE/ACM 21st International Symposium on. pp. 1–10. IEEE (2013)

[4] Brito, I., Meisels, A., Meseguer, P., Zivan, R.: Distributed Constraint Satisfaction with Partially Known Constraints. Constraints 14, 199–234 (2009)

[5] Chechetka, A., Sycara, K.: No-commitment Branch and Bound Search for Distributed Constraint Optimization. In: Proc. of the Fifth International Joint Conference on Autonomous Agents and Multiagent Systems. pp. 1427–1429. AAMAS'06, ACM, New York, NY, USA (2006)

[6] Doniec, A., Bouraqadi, N., Defoort, M., Le, V.T., Stinckwich, S.: Distributed Constraint Reasoning Applied to Multi-robot Exploration. In: Proc. of the 21st IEEE International Conference on Tools with Artificial Intelligence. pp. 159–166. ICTAI'09, IEEE Computer Society, Washington, DC, USA (2009)

[7] Ezzahir, R., Bessiere, C., Wahbi, M., Benelallam, I., Bouyakhf, E.H.: Asynchronous Inter-level Forward-checking for DisCSPs. In: Proc. of the 15th International Conference on Principles and Practice of Constraint Programming. pp. 304–318. CP'09, Springer-Verlag, Berlin, Heidelberg (2009)

[8] Floyd, R.W.: Algorithm 97: Shortest path. Commun. ACM 5(6), 345 (jun 1962)

[9] Gershman, A., Meisels, A., Zivan, R.: Asynchronous Forward-Bounding for Distributed Constraints Optimization. In: Proc. of the 2006 Conference on ECAI 2006: 17th European Conference on Artificial Intelligence. pp. 103–107. IOS Press, Amsterdam, The Netherlands, The Netherlands (2006)

[10] Gershman, A., Meisels, A., Zivan, R.: Asynchronous Forward Bounding for Distributed COPs. JAIR 34, 61–88 (2009)

[11] Grubshtein, A., Herschorn, N., Netzer, A., Rapaport, G., Yaffe, G., Meisels, A.: The Distributed Constraints (DisCo) Simulation Tool. In: Proc. of the IJCAI workshop on DCR'11. pp. 30–42. Barcelona, Catalonia, Spain (2011)

[12] Haralick, R.M., Elliott, G.L.: Increasing tree search efficiency for constraint satisfaction problems. Artif. Intel. 14(3), 263–313 (1980)

[13] Hirayama, K., Yokoo, M.: The Distributed Breakout Algorithms. Artif. Intel. 161, 89–116 (2005)

[14] Karl, H., Willig, A.: Protocols and Architectures for Wireless Sensor Networks. Wiley (2005)

[15] Kho, J., Rogers, A., Jennings, N.R.: Decentralized control of adaptive sampling in wireless sensor networks. ACM Trans. Sen. Netw. 5(3), 19:1–19:35 (Jun 2009)

[16] Kurose, J.F., Ross, K.W.: Computer Networking, 63. Addison Wesley (2013)
[17] Léauté, T., Ottens, B., Szymanek, R.: FRODO 2.0: An Open-Source Framework for Distributed Constraint Optimization. In: Proceedings of the IJCAI'09 workshop on Distributed Constraint Reasoning. pp. 160–164. Pasadena, California, USA (2009)
[18] Léauté, T., Faltings, B.: Protecting privacy through distributed computation in multi-agent decision making. J. Artif. Int. Res. 47(1), 649–695 (May 2013)
[19] Lynch, N.A.: Distributed Algorithms. Morgan Kaufmann Series (1997)
[20] Mailler, R., Lesser, V.R.: Asynchronous partial overlay: A new algorithm for solving distributed constraint satisfaction problems. JAIR 25(1), 529–576 (2006)
[21] Meisels, A., Kaplansky, E., Razgon, I., Zivan, R.: Comparing Performance of Distributed Constraints Processing Algorithms. In: Proc. of DCR'02. pp. 86–93 (2002)
[22] Meisels, A., Lavee, O.: Using additional information in DisCSP search. In: Proc. of DCR'04 (2004)
[23] Meisels, A., Zivan, R.: Asynchronous Forward-checking for DisCSPs. Constraints 12(1), 131–150 (2007)
[24] Molisch, A.F.: Wireless Communications, 2e. Wiley-Blackwell (2010)
[25] Newton, M., Pham, D., Tan, W., Portmann, M., Sattar, A.: Stochastic Local Search Based Channel Assignment in Wireless Mesh Networks. In: Proc. of the 19th International Conference on Principles and Practice of Constraint Programming (CP'13). Lecture Notes in Computer Science, vol. 8124, pp. 832–847. Springer Berlin Heidelberg (2013)
[26] Petcu, A., Faltings, B.: DPOP: A Scalable Method for Multiagent Constraint Optimization. In: Proc. of IJCAI'05. pp. 266–271 (2005)
[27] Petcu, A., Faltings, B.: A value ordering heuristic for local search in distributed resource allocation. In: Proc. of the 2004 Joint ERCIM/CoLOGNET International Conference on Recent Advances in Constraints. pp. 86–97. CSCLP'04, Springer-Verlag, Berlin, Heidelberg (2005)
[28] Pujol-Gonzalez, M., Cerquides, J., Meseguer, P., Rodríguez-Aguilar, J., Tambe, M.: Engineering the Decentralized Coordination of UAVs with Limited Communication Range. In: Advances in Artificial Intelligence, Lecture Notes in Computer Science, vol. 8109, pp. 199–208. Springer Berlin Heidelberg (2013)
[29] Silaghi, M.C., Sam-Haroud, D., Faltings, B.: Asynchronous Search With Aggregations. In: Proc. of AAAI'00/IAAI'00. pp. 917–922 (2000)
[30] Sultanik, E.A., Lass, R.N., Regli, W.C.: Dcopolis: a framework for simulating and deploying distributed constraint reasoning algorithms. In: Proc. of AAMAS'08. pp. 1667–1668 (2008)
[31] Wahbi, M., Ezzahir, R., Bessiere, C.: Asynchronous Forward Bounding Revisited. In: Proc. of the 19th International Conference on Principles and Practice of Constraint Programming (CP'13). Lecture Notes in Computer Science, vol. 8124, pp. 708–723. Springer Berlin Heidelberg (2013)
[32] Wahbi, M., Ezzahir, R., Bessiere, C., Bouyakhf, E.H.: DisChoco 2: A Platform for Distributed Constraint Reasoning. In: Proceedings of workshop on DCR'11. pp. 112–121 (2011), http://dischoco.sourceforge.net/

[33] Wahbi, M., Ezzahir, R., Bessiere, C., Bouyakhf, E.H.: Nogood-Based Asynchronous Forward-Checking Algorithms. Constraints 18(3), 404–433 (2013)
[34] Wallace, R.J., Freuder, E.C.: Constraint-based reasoning and privacy/efficiency tradeoffs in multi-agent problem solving. Artif. Intel. 161, 209–228 (2005)
[35] Warshall, S.: A theorem on boolean matrices. J. ACM 9(1), 11–12 (jan 1962)
[36] Wu, X., Brown, K.N., Sreenan, C.J.: Data pre-forwarding for opportunistic data collection in wireless sensor networks. In: Networked Sensing Systems (INSS), 2012 Ninth International Conference on. pp. 1–8. IEEE (2012)
[37] Yokoo, M.: Algorithms for distributed constraint satisfaction problems: A review. Journal of AAMAS 3(2), 185–207 (2000)
[38] Yokoo, M., Durfee, E.H., Ishida, T., Kuwabara, K.: Distributed Constraint Satisfaction for Formalizing Distributed Problem Solving. In: Proc. of 12th IEEE International Conference on Distributed Computing Systems. pp. 614–621 (1992)
[39] Zhang, W., Wang, G., Xing, Z., Wittenburg, L.: Distributed stochastic search and distributed breakout: properties, comparison and applications to constraint optimization problems in sensor networks. Artif. Intel. 161, 55–87 (2005)
[40] Zivan, R., Meisels, A.: Dynamic Ordering for Asynchronous Backtracking on DisCSPs. Constraints 11(2-3), 179–197 (2006)
[41] Zivan, R., Meisels, A.: Message delay and DisCSP search algorithms. Annals of Mathematics and Artificial Intelligence 46(4), 415–439 (2006)