Constraint Acquisition System For Distributed Constraint Problem With Two-Agents

Hajar Ait Addi, Redouane Ezzahir

LISTI/ENSA, University of Ibn Zohr,
Morocco, PH-+212 6 03 74 08 59
hajar.aitaddi.edu@uiz.ac.ma

LISTI/ENSA, University of Ibn Zohr,
Morocco, PH-+212 6 77 03 58 16
r.ezzahir@uiz.ac.ma

Abstract: Constraint programming (CP) is a powerful paradigm for solving and modeling combinatorial problems. Nevertheless, building a CP model requires some expertise in constraint programming. The users find it difficult to articulate their constraints, while they are able to recognize examples of where a constraint has to be satisfied or violated. Several constraint acquisition systems have been introduced to take an active role in acquiring the user's constraints. However, until recently, no such system existed for distributed constraint problem (DCP). In this paper, we attempt to present a new algorithm of constraint acquisition for distributed constraint problem with two-agents (DisCP2A). We propose to improve the recent QuAcq system to ac- quire automatically such problem, this lead to a new system called Dis-QuAcq. We apply our basic approach in context of a distributed problem involving the acquisition of SensorDCP with two-mobile constraints. Finally, we conclude the paper.

Keywords: Constraint programming · Constraint acquisition · Distributed Constraint · Agent

1. Introduction

Constraint programming (CP) is used to model and solve complex combinatorial problems. However, building a CP model requires some expertise in constraint programming. In this situation, several approaches have been proposed to take an active role in acquiring constraints. The matchmaker agent [6] proposed by Freuder and Wallance. When the system proposes an incorrect solution, the agent asks the user to communicate a new constraint that explains why she considers a proposed solution as a wrong one. Lallouet et al. introduced a system based on inductive logic programming [4]. Beldiceanu and Simonis have proposed MODELSEEKER, a system devoted to problems with regular structures and based on the global constraint catalog [3]. Bessiere et al. proposed CONACQ, which interactively proposes to the user member- ship queries (i.e., complete examples) to be classified by the user [5, 7]. Bessiere et al. proposed QuAcq (for Quick Acquisition), an active learning system that is able to ask the user to classify partial queries [1, 8]. If the user says yes, **QuAcq** removes from the search space all constraints violated by the positive example. If the user says no, **QuAcq** finds the scope of one of the violated constraints.

All these systems are centralized. Nevertheless, most problems in the real-world application are naturally distributed. The information of a problem aren't possible to be gathered in one site, so we must distribute this information between participants (Agents). To deal with this type of problem, the scientific researchers have developed the context of networked distributed systems to model combinatorial problems arising in distributed multi-agent environments. Such context is called distributed constraint programming (DisCP). There is a rich set of distributed applications for which DisCP formulation is useful. For instance, distributed sensor networks (**SensorDCP**) [9] is a naturally distributed problem. Another example is distributed planning problems.

In this paper we introduce **Dis-QuAcq** a new system to model problems formulated in distributed fashion with two-agents. **Dis-QuAcq** algorithm inherits its basic performance from **QuAcq** system. Section 2 presents the necessary background on distributed constraint programming with two-agents and constraint acquisition. Section 3 describes **Dis-QuAcq** algorithm for distributed quick acquisition. We describe **SensorDCP** benchmark, and we model this problem using our approach **Dis-QuAcq** in section 4. Section 5 concludes the paper.

2. Background

2.1 Distributed Constraint Programming with two-agents

A distributed constraint network with tow-agents is a quintuple (X, D, A, φ), where:

- $X = \{x_1, \ldots, x_n\}$, a set on n variables.
- $D = \{D(x_1), \ldots, D(x_n)\}$, where $D(x_i) \subset Z$ is the finite set of values for x_n.
- $C = \{c_{ij} | x_i, x_j \in X\}$ is a set of constraint of the global problem.
- $A = \{A_1, A_2\}$ is a set of two agents.
- $\varphi : X \rightarrow A$, that matches each variable to an agent.

The set C is divided into two subsets: set of intra-agent C_{intra} and set of inter- agent C_{inter}.

- $C_{intra} = \{c_{ij} | \varphi(x_i) = \varphi(x_j)\}$ the variables x_i and x_i belong to the same agent.
- $C_{inter} = \{c_{ij} | \varphi(x_i) \neq \varphi(x_j)\}$ the variables x_i and x_i belong to different agents.

2.2 Constraint acquisition

The constraint acquisition process can be seen as interplay between the $user_A$ (related to an agent A) and the learner.

$User_A$ and learner need to share a vocabulary to communicate. We suppose this vocabulary is a set (X_A,D_A), where X_A, D_A variables, and domains controlled by the agent A. A constraint c_Y is defined by a sequence Y of variables of X_A, called the constraint scope, and the relation c over D of arity |Y|. An assignment e_Y on a set of variables $Y \subset X_A$ violates a constraint c_Z (or e_Y is rejected by c_Z) if $Z \subset Y$ and the projection e_Z of e_Y on the variables in Z is not in c. A constraint network is a set C_A of constraints on the vocabulary (X_A,D_A). An assignment on X_A is a solution of C_A if and only if it does not violate any constraint in C_A. sol(C_A) represents the set of solutions of C_A.

In addition to the vocabulary, the learner owns a language Γ of relations, from which it can build constraints on specified sets of variables.

Adapting terms from machine learning, the constraint bias, denoted by B_A, is a set of constraints built from the constraint language Γ on the vocabulary (X_A,D_A), from which the learner builds the constraint network. The target network is a network C_{T_A} such that for any example $e \in D^{X_A} = \prod_{x_j \in X_A} D(x_i)$, e is a solution of C_{T_A} if and only if e is a solution of the problem that the $user_A$ has in mind.

A membership query $Ask_A(e)$ is a classification question asked to the $user_A$, where e is a complete assignment in D^{X_A}. The answer to $Ask_A(e)$ is yes if and only if $e \in sol(C_{T_A})$. A partial query $Ask_A(e_Y)$ with $Y \subset X_A$, is a classification question asked to the$user_A$, where e_Y is a partial assignment in $D^Y = \prod_{x_j \in Y} D(x_i)$. The answer $Ask_A(e_Y)$ is yes if and only if e_Y does not violate any constraint in C_{T_A}. A classified assignment e_Y is called a positive or negative example depending on whether $Ask_A(e_Y)$ is yes or no. For any assignment e_Y on Y, $\kappa_B(e_Y)$ denotes the set of all constraints in B_A rejecting e_Y .

3. Dis-QuAcq Algorithm

We propose **Dis-QuAcq**, a novel algorithm for distributed constraint acquisition problem with two agents (A_1, A_2). To learn intra-constraints C_{intra} related to agent A_1 and agent A_2, the learner will separately interact with the $user_{A_1}$ and the $user_{A_2}$. For inter-constraints C_{inter}. The learner needs a new type of membership query to ask to classify queries.

Definition: A membership query Ask_{multi} (e) (e partial or complete) is a classification question asked by the learner to the users: $user_{A_1}$and $user_{A_2}$.
Ask_{multi}(e) = ($AskM_{A_1}$ (e), $AskM_{A_2}$ (e)), where $AskM_A$ (e) is a new membership query that ask the user to classify a query with respect to inter-constraints C_{inter}. A query is positive if and only if e doesn't violate any inter-constraints of A.

To learn intra-constraints of an agent A, we will use exactly **QuAcq** algorithm. For inter-constraints, we will rewrite **QuAcq** to be compatible with the fact of asking two users. This new version of **QuAcq** is called $\boldsymbol{QuAc_{multi}}$ for multi-users.

3.1 Description of Dis-QuAcq

Dis-QuAcq takes as input a bias B_{A_1} on the vocabulary (X_{A_1},D_{A_1}), B_{A_2} on the vocabulary (X_{A_2},D_{A_2}), and B on the vocabulary (X, D). The bias B_{A_1} (B_{A_2}) contains all possible intra-constraints related to the variables of A_1(A_2) that can be generated from the relations in a given language. The bias B contains all inter-constraints related to X that can be generated from the relations in a given language.

In lines 1 and 2, Dis-QuAcq learns intra-constraints of agent A_1 and agent A_2 using **QuAcq** algorithm. Next, Dis-QuAcq calls $\boldsymbol{QuAc_{multi}}$ to learn inter-constraints of A_1 and agent A_2 (line 3).

Algorithm 1: Dis-QuAcq

1 **QuAcq**(X_{A_1}, D_{A_1},B_{A_1}) ;
2 **QuAcq**(X_{A_2}, D_{A_2},B_{A_2}) ;
3 **QuAcq**$_{multi}$(X, D,B) ;

Figure 1: Dis-QuAcq Algorithm

3.2 Description of QuAcq

In this section, we present **QuAcq** algorithm. **QuAcq** algorithm differs from the basic version presented in [1] in the fact that the functions ***FindC*** and *FindScope* are indexed by the name of the user who interacts with the learner. We describe now the performance of **QuAcq. QuAcq** initializes $C_{L_A}$to the empty set (line 1). If C_{L_A} is unsatisfied, we return collapse because the learned network is inconsistent. Next, we pick up an assignment e that satisfies C_{L_A} and rejects at least one constraint from the bias BA (line 3). If such example doesn't exist, we have reached the convergence state (line 4). Otherwise, **QuAcq** asks the user to classify the selected example. If the example is positive (line 5), we update the bias BA by removing all constraints that reject our example in line 5. If the example is negative, we call $\boldsymbol{FindC_A}$ and $\boldsymbol{FindScope_A}$ to discover the violated constraint c (line 7) . We have changed the nomination of ***FindC*** and *FindScope* functions that appear in [1], to suit the concept of two users. These two functions interacts with the user using the membership query Ask_A(e). If the searched constraint is founded, we update C_{L_A} by adding the returned constraint c (line 23). Otherwise, we return collapse as we could not find in B_A a constraint rejecting the negative example (line 8).

Algorithm 2: QuAcq(X_A, D_A,B_A)

1 $C_{L_A} \leftarrow \emptyset$;
2 **if** *sol* $(C_{L_A}) = \emptyset$ **then return**"Collapse";
3 choose e in D^{X_A} accepted by C_{L_A} and rejected by B_A ;
4 **if** *sol* (e) = *nil* **then return**"Convergence on C_{L_A}" ;
5 **if** $Ask_A(e)$= yes **then** $B_A \leftarrow B_A \setminus \kappa_{B_A}(e)$;
6 **else**
7 $c \leftarrow$ FindC$_A$(e,FindScope$_A$(e,$\emptyset$,X_A,$false$)) ;
8 **if** c=*nil* **then return**"Collapse" ;
9 **else** $C_{L_A} \leftarrow C_{L_A} \cup \{c\}$;

Figure 2: QuAcq Algorithm

3.3 Description of $QuAcq_{multi}$

Algorithm 3: QuAcq$_{multi}$

1 $C_L \leftarrow \emptyset$;
2 **while** *true* **do**
3   **if** *sol* $(C_L) = \emptyset$ **then return**"Collapse";
4   choose e in D^X accepted by C_L and rejected by B ;
5   **if** *sol* $(e) = nil$ **then return**"Convergence on C_L";
6   **if** $Ask_{multi}(e) =$ (yes,yes) **then** $B \leftarrow B \setminus \kappa_B(e)$;
7   **else**
8   **if** $Ask_{multi}(e) =$ (yes,no) **then**
9             $c \leftarrow \mathtt{FindC}_{A_2}(e, \mathtt{FindScope}_{A_2}(e, \emptyset, X, false))$;
10   **if** c=*nil* **then return**"Collapse";
11   **else** $C_L \leftarrow C_L \cup \{c\}$;
12   **if** $Ask_{multi}(e) =$ (no,yes) **then**
13             $c \leftarrow \mathtt{FindC}_{A_1}(e, \mathtt{FindScope}_{A_1}(e, \emptyset, X, false))$;
14   **if** c=*nil* **then return**"Collapse";
15   **else** $C_L \leftarrow C_L \cup \{c\}$;
16   **else**
17             $c \leftarrow \mathtt{FindC}_{multi}(e, \mathtt{FindScope}_{multi}(e, \emptyset, X, false))$;
18   **if** c=*nil* **then return**"Collapse";
19             $c' \leftarrow \mathtt{FindC}_{multi}(e, \mathtt{FindScope}_{multi}(e, \emptyset, X, false))$;
20   **if** $c' = nil$ **then return**"Collapse";
21   **else**
22                 $C_L \leftarrow C_L \cup \{c, c'\}$;
23                 $B \leftarrow B \setminus \{c, c'\}$;

We present $QuAcq_{multi}$ a novel algorithm that interacts with two users (A_1, A_2) to learn a set of inter-constraints. $QuAcq_{multi}$ takes as input a bias B on the vocabulary (X, D). In line 1, we initialize C_L to the empty set. If C_{LA} is unsatisfied (line 3), we return collapse because the learned network is inconsistent. Next, we attempt to generate a query e that accepts CL and rejects at least a constraint from B. If such query doesn't found, means that $QuAcq_{multi}$ have reached convergence. Otherwise, we should ask the users to classify the query e. If both users classify the query as positive (line 6), we removes from the bias B all constraints that reject the query. If only one user A_i(i∈{1,2}) who classify the query as negative (lines 8 and 12), we call the functions $FindC_{A_i}$ and $FindScope_{A_i}$ to return the violated constraint (lines 9, 13). These two functions interacts with the user using the membership query $AskM_A$(e), because $QuAcq_{multi}$interests only on inter-constraints. If such constraint exists, we added it to the learned network C_L . If not, we reach a col lapse state. If both users classify the query as negative (line 16), we call the functions $FindC_{multi_{A_1,}}$ and $FindScope_{multi_{A_1,}}$ to interact with the user A_1, to find the violated inter-constraint c of user A_1,. If such constraint doesn't exist, we return collapse. Next, we call $FindC_{multi_{A_2}}$ and $FindScope_{multi_{A_2}}$ to discover the violated inter-constraint c' of user A_2. If such constraint doesn't found, we have another condition of collapsing. Otherwise, we update the learned network by adding the returned constraints c and c' .

The functions $\boldsymbol{FindC_{multi_{A,}}}$and $\boldsymbol{FindScope_{multi_A}}$ differ from $\boldsymbol{FindC_A}$ and $\boldsymbol{FindScope_A}$ in deleting the lines where $\boldsymbol{QuAcq}_{multi}$ removes constraints from the bias B. Removing constraints from the bias B necessitates that both users classify the query as positive. In algorithm 4, the deleted lines from $\boldsymbol{FindScope_A}$ function are represented in gray, and the new ones are in yellow.

Algorithm 4: $\mathtt{FindScope}\ _{multi_A}(e, R, Y, ask\_query)$

begin
if *ask_query* **then**
    if $AskM_A(e[R]) =$ yes **then** $B_A \leftarrow B_A \setminus \kappa_{B_A}(e)$;
    else return $\emptyset$;
if *ask_query* **then**
    if $AskM_A(e[R]) =$ no **then return** $\emptyset$;
if $|Y| = 1$ **then return** Y;
split Y into $< Y_1, Y_2 > such that |Y_1| = [|Y|/2]$;
$S_1 \leftarrow \mathtt{FindScope}_{multi_A}(e, R \cup Y_1, Y_2, true)$;
$S_2 \leftarrow \mathtt{FindScope}_{multi_A}(e, R \cup S_1, Y_1, (S_1 \neq \emptyset)$;
return $S_1 \cup S_2$;

Figure 3: $\boldsymbol{FindScope_{multi_A}}$ Algorithm

The table 1 summarizes the change made on the basic version of $\boldsymbol{FindC}$ and $\boldsymbol{FindScope}$ functions presented in [8].

Table 1: The performance of $\boldsymbol{FindScope}$ and $\boldsymbol{FindC}$

Function	Algorithm	members hip query	change made
$\boldsymbol{FindC_A}$ $\boldsymbol{FindScope_A}$	$QuAcq$	Ask_A	-
$\boldsymbol{FindC_A}$ $\boldsymbol{FindScope_A}$	$\boldsymbol{QuAcq}_{multi}$	$AskM_A$	-
$\boldsymbol{FindC_{multi_{A,}}}$ $\boldsymbol{FindScope_{multi_A}}$	$\boldsymbol{QuAcq}_{multi}$	$AskM_A$	Deleting the lines that remove constraints from B

4. Application

4.1 Benchmark Problem: SensorDCP

SensorDCP is naturally distributed benchmark that appears in the context of net- worked distributed system. In this problem, we have multiple sensors $S = \{s_1, \ldots, s_n\}$, and multiple mobile $T = \{t_1, \ldots, t_n\}$, which are to be followed by the sensors.
The goal is:

- each mobile should be tracked by three senors;
- Each sensor can track at most one mobile;

The solution to this problem is an allocation of three distinct sensors to each mobile. This allocation must satisfy:

- Visibility constraints: defines the set of sensors of a mobile that can possibly detect it;
- Compatibility constraints: the sensors of a mobile must satisfy compatibility relation with each other.

DisCP encodes the **SensorDCP** problem as follow:

- Each mobile represents a different agent;
- Each agent controls three variables, one for each sensor that we must assign to the corresponding mobile;

- The value domain of each variable is the set of sensors that can detect the corresponding mobile;
- The intra-constraints between the variables of an agent are that the three sensors allocated to the mobile must be distinct and compatible.
- The inter-constraints between variables of different agents are that a given sensor will be chosen by one agent at most.

For our experiment, we choose to apply **Dis-QuAcq** on **SensorDCP** with two mobiles to acquire the constraints. **SensorDCP** with two-mobiles and n sensors $\{s_1, \ldots, s_n\}$: This is encoded with two agent A_1 et A_2 with three variables for each agent.
The variables of A_1 are $X_{A_1} = \{x_1, x_2, x_3\}$ and $X_{A_2} = \{x_4, x_5, x_6\}$ are the variables of A_2. The global problem has 6 variables of domain size of n. We feed B_{A_1}, B_{A_2} with unary and binary constraints from the language $\Gamma = \{=_i, \neq_i, =, \neq\}$ (i ∈ 1 . . . n). And we feed B with binary constraints from the language Γ'={=, ≠}.

4.2 Result

Figure 4 reports the results of **Dis-QuAcq**. totT is the total time of the learning process, MT (q) the maximum waiting time between two queries, and #q the total num- ber of asked queries.

The experimental results obtained on **SensorDCP** with two mobile shows that our framework learns all constraints. In figure 4, we notice that the number of query needed to learn intra-constraints is the same for both users. The reason is that both agents have similar intra-constraints. For the number of query in $QuAcq_{multi}$ is bigger than the number needed in **QuAcq**. As both users have the same inter-constraints, when one of them classifies an example as negative, the other one also classifies this example as negative. So $QuAcq_{multi}$ calls the functions $FindC_{multi_A}$ and $FindScope_{multi_A}$.

Algorithm Dis-QuAcq	$totT$ (sec)	$MT(q)$ (sec)	$\#q$
Number of sensors	n=6		
QuAcq$(X_{A_1}, D_{A_1}, B_{A_1})$	0.18	0.06	44
QuAcq$(X_{A_2}, D_{A_2}, B_{A_2})$	0.18	0.06	44
QuAcq$_{multi}(X, D, B)$	0.16	0.06	172
Number of sensors	n=9		
QuAcq$(X_{A_1}, D_{A_1}, B_{A_1})$	0.20	0.05	62
QuAcq$(X_{A_2}, D_{A_2}, B_{A_2})$	0.20	0.05	62
QuAcq$_{multi}(X, D, B)$	0.17	0.06	176

Figure 4: Results of **Dis-QuAcq** learning **SensorDCP** with two-mobiles and n sensors

5. Conclusion

In this paper, we have outlined a model **Dis-QuAcq** of learning constraints for distributed problem with two agents based on functions from **QuAcq** algorithm. We have applied our approach in the context of a simple example involving the acquisition of **SensorDCP** with tow agents constraints.

References

[1] Christian Bessiere, Remi Coletta, Emmanuel Hebrard, George Katsirelos, Nadjib Lazaar, Nina Naro- dytska, Claude-Guy Quimper, and Toby Walsh. Constraint acquisition via partial queries. In Proceedings of the 23rd International Joint Conference on Artificial Intelligence, IJCAI 2013, pages 475–481, Beijing, China, 2013.

[2] Carine Lallemand and Guillaume Gronier. Enhancing user experience during waiting time in hci: Contributions of cognitive psychology. In Proceedings of the Designing Interactive Systems Conference, DIS '12, pages 751–760, New York, NY, USA, 2012. ACM.

[3] Nicolas Beldiceanu and Helmut Simonis. A model seeker: Extracting global constraint models from positive examples. In Proceedings of the 18th International Conference on Principles and Practice of Constraint Programming, CP 2012, volume 7514 of Lecture Notes in Computer Science, pages 141– 157, Que´bec City, QC, Canada, 2012. Springer.

[4] Arnaud Lallouet, Matthieu Lopez, Lionel Martin, and Christel Vrain. On learning constraint problems. In Proceedings of the 22nd IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2010, pages 45–52, Arras, France, 2010.

[5] Christian Bessiere, Remi Coletta, Barry O'Sullivan, and Mathias Paulin. Query-driven constraint acquisition. In Proceedings of the 20th International Joint Conference on Artificial Intelligence, IJCAI 2007, pages 50–55, Hyderabad, India, 2007.

[6] Eugene C. Freuder and Richard J. Wallace. Suggestion strategies for constraint-based matchmaker agents. International Journal on Artificial Intelligence Tools, 11(1):3–18, 2002.

[7] Christian Bessiere, Nadjib Lazaar, Frédéric Koriche, and Barry O'Sullivan. Constraint acquisition. Artificial Intelligence, page In Press, 2017.

[8] Christian Bessiere, Abderrazak Daoudi, Emmanuel Hebrard, George Katsirelos, Nadjib Lazaar, Younes Mechqrane, Nina Narodytska, Claude-Guy Quimper, and Toby Walsh. New approaches to constraint acquisition. In Data Mining and Constraint Programming - Foundations of a Cross-Disciplinary

Approach, volume 10101 of Lecture Notes in Computer Science, pages 51–76. Springer, 2016

[9] R. Béjar, C. Domshlak, C. Fernández, C. Gomes, B. Krishnamachari, B. Selman, and M. Valls (2005), "Sensor networks and distributed csp : communication, computation and complexity". Artificial Intelligence, 161 :117147. Cited on page 61