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 acquire 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, ..., x_n\}, a set on n variables.
- D = \{D(x_1), ..., D(x_n)\}, where D(x_i) ⊂ Z is the finite set of values for x_n.
- C = \{c_{ij}\} the variables \{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.
- φ : X → 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}|φ(x_i) = φ(x_j)\} the variables x_i and x_j belong to the same agent.
- C_{inter} = \{c_{ij}|φ(x_i) ≠ φ(x_j)\} the variables x_i and x_j 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 \( \mathcal{c}_Y \) is defined by a sequence \( Y \) of variables of \( X_A \), called the constraint scope, and the relation \( c \) of \( D \) of \( Y \) [1]. An assignment \( \mathcal{e}_Y \) on a set of variables \( Y \subset X_A \) violates a constraint \( \mathcal{c}_Z \) (or \( \mathcal{c}_Z \) is rejected by \( \mathcal{e}_Y \)) if \( Z \subset Y \) and the projection \( \mathcal{e}_Z \) of \( \mathcal{e}_Y \) on the variables in \( Z \) is not in \( c \). A constraint network is a set \( \mathcal{C}_A \) of constraints on the vocabulary \((X_A, D_A)\). An assignment on \( X_A \) is a solution of \( \mathcal{C}_A \) if and only if it does not violate any constraint in \( \mathcal{C}_A \). \( \mathcal{C}_A \) represents the set of solutions of \( \mathcal{C}_A \).

In addition to the vocabulary, the learner owns a language \( \Gamma \) 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 \( \Gamma \) on the vocabulary \((X_A, D_A)\), from which the learner builds the constraint network. The target network is a network \( C_{TA} \) such that for any example \( e \in D^A = \prod_{x \in X_A} D(x) \), \( e \) is a solution of \( C_{TA} \) if and only if \( e \) is a solution of the problem that the user \( A \) has in mind.

A membership query \( \text{Ask}_A(e) \) is a classification question asked to the user \( A \), where \( e \) is a complete assignment in \( D^A \). The answer to \( \text{Ask}_A(e) \) is yes if and only if \( e \in \text{sol}(C_{TA}) \). A partial query \( \text{Ask}_A(e_Y) \) with \( Y \subset X_A \) is a classification question asked to the user \( A \), where \( e_Y \) is a partial assignment in \( D^Y = \prod_{x \in Y} D(x) \). The answer \( \text{Ask}_A(e_Y) \) is yes if and only if \( e_Y \) does not violate any constraint in \( C_{TA} \). A classified assignment \( e_Y \) is called a positive or negative example depending on whether \( \text{Ask}_A(e_Y) \) is yes or no. For any assignment \( e_Y \) on \( Y \), \( \kappa_\Gamma(e_Y) \) 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 \( C_{\text{intra}} \) 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 \( C_{\text{inter}} \). The learner needs a new type of membership query to ask to classify queries.

**Definition:** A membership query \( \text{Ask}_{\text{multi}}(e) \) (e partial or complete) is a classification question asked by the learner to the users: \( \text{user}_{A_1} \) and \( \text{user}_{A_2} \).

\[ \text{Ask}_{\text{multi}}(e) = \left( \text{AskM}_A(e), \text{AskM}_A(e) \right), \]

where \( \text{AskM}_A(e) \) is a new membership query that asks the user to classify a query with respect to inter-constraints \( C_{\text{inter}} \). 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 **QuAcq_{multi}** for multi-users.

#### 3.1 Description of Dis-QuAcq

**Dis-QuAcq** takes as input a bias \( B_A \) on the vocabulary \((X_A, D_A)\), \( B_A \) on the vocabulary \((X_A, D_A)\), and \( B \) on the vocabulary \((X, D)\). The bias \( B_A \) \((B_A)\) 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 **QuAcq_{multi}** to learn inter-constraints of \( A_1 \) and agent \( A_2 \) (line 3).

#### 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 \( \text{FindC} \) and \( \text{FindScope} \) are indexed by the name of the user who interacts with the learner. We describe now the performance of **QuAcq**. **QuAcq** initializes \( C_{\text{LA}} \) to the empty set (line 1). If \( C_{\text{LA}} \) is unsatisfied, we return collapse because the learned network is inconsistent. Next, we pick up an assignment \( e \) that satisfies \( C_{\text{LA}} \) 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 \( \text{FindC}_A \) and \( \text{FindScope}_A \) to discover the violated constraint \( c \) (line 7). We have changed the nomination of \( \text{FindC} \) and \( \text{FindScope} \) functions that appear in [1], to suit the concept of two users. These two functions interacts with the user using the membership query \( \text{Ask}_A(e) \). If the searched constraint is founded, we update \( C_{\text{LA}} \) by adding the returned constraint \( c \) (line 23). Otherwise, we return collapse as we could not find in \( BA \) a constraint rejecting the negative example (line 8).

![Figure 1: Dis-QuAcq Algorithm](image1)

![Figure 2: QuAcq Algorithm](image2)
3.3 Description of $QuAcq_{multi}$

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 $C_L$ 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 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 $AskM_{A_i}(e)$, 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 collapse state. If both users classify the query as negative (line 16), we call the functions $FindC_{multi_{A_i}}$ and $FindScope_{multi_{A_i}}$ 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 $FindC_{multi_{A_i}}$ and $FindScope_{multi_{A_i}}$ differ from $FindC_{A_i}$ and $FindScope_{A_i}$ in deleting the lines where $QuAcq_{multi}$ 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 $FindScope_{A_i}$ function are represented in gray, and the new ones are in yellow.

Algorithm 4: $FindScope_{multi_{A_i}}(e,R,Y,ask\_query)$

begin
1. if ask\_query then
2. if $AskM_{A_i}(e|R) = yes$ then $B_A \leftarrow B_A \setminus S_{B_A}(e)$;
else return $\emptyset$;
end
if $ask\_query$ then
3. if $|Y| = 1$ then return $Y$;
4. split $Y$ into $Y_1, Y_2$ such that $|Y_1| = |Y_2|$;
5. $S_1 \leftarrow FindScope_{multi_{A_i}}(e,R \cup Y_1, Y_2, true)$;
6. $S_2 \leftarrow FindScope_{multi_{A_i}}(e,R \cup Y_1, Y_2, false)$;
7. return $S_1 \cup S_2$;
end

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

Table 1: The performance of $FindScope$ and $FindC$

<table>
<thead>
<tr>
<th>Function</th>
<th>Algorithm</th>
<th>membership query</th>
<th>change made</th>
</tr>
</thead>
<tbody>
<tr>
<td>FindC_{A_i}</td>
<td>QuAcq</td>
<td>Ask_{A_i}</td>
<td>-</td>
</tr>
<tr>
<td>FindScope_{A_i}</td>
<td>QuAcq_{multi}</td>
<td>AskM_{A_i}</td>
<td>-</td>
</tr>
<tr>
<td>FindC_{multi_{A_i}}</td>
<td>QuAcq_{multi}</td>
<td>AskM_{A_i}</td>
<td>Deleting the lines that remove constraints from $B$</td>
</tr>
</tbody>
</table>

4. Application

4.1 Benchmark Problem: SensorDCP

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

The goal is:
- each mobile should be tracked by three sensors;
- 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 \( \Gamma = \{=, \neq, =, \neq\} \) ( \( i \in 1 \ldots n \) ). And we feed B with binary constraints from the language \( \Gamma' = \{=, \neq\} \).

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 mobiles 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\(_{multi}\) and FindScope\(_{multi}\).

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


