

# An approach to symmetry breaking in DCR

IJCAI '09, DCR workshop

Xavier Olive and Hiroshi Nakashima

Graduate School of Informatics  
Kyoto University

July 13th, 2009

Introduction

Definitions & algorithms

Symmetry breaking

Results

Conclusion & future works

# Introduction

---

- ▶ (Distributed) constraint reasoning is a powerful paradigm adapted for solving various engineering problems.
- ▶ Symmetry breaking can show some improvements on centralized constraint programming.
- ▶ Though the idea of symmetry breaking may sound incompatible with the distribution of data, ...
- ▶ ... we suggest a way to take some symmetries into account.

Introduction

Definitions & algorithms

Definitions

Algorithms

Symmetry breaking

Results

Conclusion & future works

## Constraint solving problem

A constraint solving problem is based on:

- ▶ a finite set of variables  $\mathcal{X} = \{x_1, \dots, x_k\}$ ,
- ▶ a finite set of domains  $\mathcal{D} = \{d_1, \dots, d_k\}$ ,
- ▶ a finite set of constraints  $\mathcal{C}$ , each  $c \in \mathcal{C}$  being a subset of  $d_1 \times \dots \times d_k$ .

## Solvability

- ▶ If the resulting constraint  $\mathcal{C}$  is not empty, the problem is solved, and the resulting subset is the set of solutions.

## Distributed constraint solving problem

A **distributed** constraint solving problem is based on:

- ▶ a finite set of variables  $\mathcal{X} = \{x_1, \dots, x_k\}$ ,
- ▶ a finite set of domains  $\mathcal{D} = \{d_1, \dots, d_k\}$ ,
- ▶ a finite set of constraints  $\mathcal{C}$ , each  $c \in \mathcal{C}$  being a subset of  $d_1 \times \dots \times d_k$ .
- ▶ a finite set of not necessarily different agents  $\mathcal{A} = \{a_1, \dots, a_k\}$

## Solvability

- ▶ If the resulting constraint  $\mathcal{C}$  is not empty, the problem is solved, and the resulting subset is the set of solutions.

Introduction

Definitions & algorithms

Definitions

Algorithms

Symmetry breaking

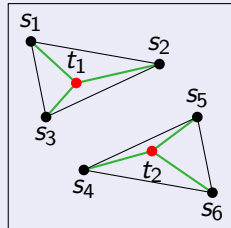
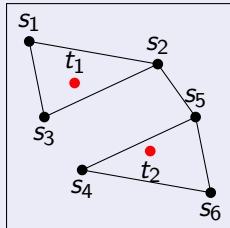
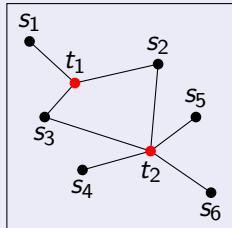
Results

Conclusion & future works

## SensorDCSP benchmark

- ▶ multiple sensors ( $s_i$ ) and multiple mobiles ( $t_i$ ).
- ▶ allocate 3 distinct sensors to track each mobile node, subject to visibility and compatibility constraints.

## Visibility, compatibility and solution



## Remarks

- ▶ Real world application, context of network distributed systems.
- ▶ The problem is NP-complete: partitioning a graph into cliques of size 3.

## Simulator

- ▶ We developed a simulator with fixed sensors and mobiles moving over a map.
- ▶ Part of the communication and resolution framework comes from FRODO framework developed by EPFL/LIA.

## Algorithms

- ▶ First algorithms (ABT): communicate changes to neighbours, whose number keeps increasing.
  - ▶ Better methods (ADOPT, DPOP) have been developed taking into account which variables are constrained together. DFS structures.
- 
- ▶ We kept in mind that DFS structures give better results for making some choices (later in the presentation).

Introduction

Definitions & algorithms

Symmetry breaking

Introduction to CSP symmetries

Adapting to distributed problems

Results

Conclusion & future works



## Symmetry

A symmetry for a CSP is:

- ▶ a **mapping of the CSP onto itself** that preserves its solution.
- ▶ a **permutation** over the values or variables that leaves the whole set of constraints unchanged.

## Symmetry

A symmetry for a CSP is:

- ▶ a **mapping of the CSP onto itself** that preserves its solution.
- ▶ a **permutation** over the values or variables that leaves the whole set of constraints unchanged.

## Examples of symmetry

- ▶ on a single equation:

$$x + y = z \xrightarrow{x \rightleftharpoons y} y + x = z$$

## Symmetry

A symmetry for a CSP is:

- ▶ a **mapping of the CSP onto itself** that preserves its solution.
- ▶ a **permutation** over the values or variables that leaves the whole set of constraints unchanged.

## Examples of symmetry

- ▶ on a single equation:

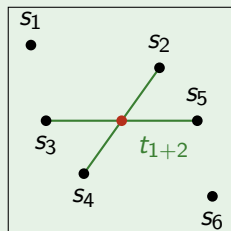
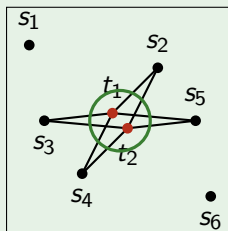
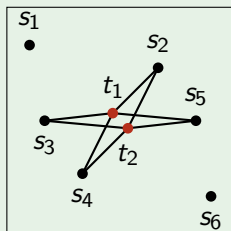
$$x + y = z \xrightarrow{x \rightleftharpoons y} y + x = z$$

- ▶ on a system of equations:

$$\begin{array}{l} x + 2y = z \\ t + 2y = z \end{array} \xrightarrow{x \rightleftharpoons t} \begin{array}{l} t + 2y = z \\ x + 2y = z \end{array}$$

## Symmetries in SensorDCSP

- If we have two mobiles “close” to each other, we can see them as one.



## Exploiting symmetry

A symmetry over a CSP can:

- ▶ be exploited during search (avoid revisiting equivalent states),
- ▶ induce extra constraints which simplify the problem,
- ▶ induce a reformulation of the problem into a smaller one (preprocessing).

## Exploiting symmetry

A symmetry over a CSP can:

- ▶ be exploited during search (avoid revisiting equivalent states),
- ▶ induce extra constraints which simplify the problem,
- ▶ **induce a reformulation of the problem into a smaller one** (preprocessing).

## Detecting symmetry

- ▶ Elements of group theory let us find **all the symmetries**
- ▶ However, the detection time may not be compensated by the time we save solving a simplified problem.

Introduction

Definitions & algorithms

**Symmetry breaking**

Introduction to CSP symmetries

Adapting to distributed problems

Results

Conclusion & future works

## Issues due to distribution of data

- ▶ The definition of a symmetry itself requires to know the whole problem; yet, we don't want to centralize the data.
- ▶ Each agent knows a subproblem, so :
  - ▶ we can search the symmetries on each subproblem, and
  - ▶ check their validity with the neighbouring agents.

## Issues due to distribution of data

- ▶ The definition of a symmetry itself requires to know the whole problem; yet, we don't want to centralize the data.
- ▶ Each agent knows a subproblem, so :
  - ▶ we can search the symmetries on each subproblem, and
  - ▶ check their validity with the neighbouring agents.

## For example

- ▶  $a_1 : x_1, z_1$  and  $a_2 : y_2, t_2$  as agents/variables
- ▶  $x_1 + y_2 = 2 \cdot z_1$  and  $|x_1 - y_2| = t_2$  as constraints
- ▶  $a_1 : x_1 \rightleftharpoons y_2$  is a  **$a_1$ -loc. sym.**, send  $x_1 \rightleftharpoons y_2$  to  $a_2$
- ▶  $a_2 : \text{recv } x_1 \rightleftharpoons y_2, x_1 \rightleftharpoons y_2$  is a  **$a_2$ -loc. sym.**
- ▶  $x_1 \rightleftharpoons y_2$  is a **global symmetry**

## Proposition of algorithm (1/2)

- ▶ We set a priority order on the agents.

## Proposition of algorithm (1/2)

- ▶ We set a priority order on the agents.
- ▶  $\mathcal{A}_i$  is the set of variables of lower priority who share a constraint with  $a_i$ .

$$a_0, a_1, a_2, \dots, a_i, \underbrace{\dots a_j, \dots a_k, \dots a_n}_{\mathcal{A}_i = \{a_j, a_n\}}$$

## Proposition of algorithm (1/2)

- ▶ We set a priority order on the agents.
- ▶  $\mathcal{A}_i$  is the set of variables of lower priority who share a constraint with  $a_i$ .

$$a_0, a_1, a_2, \dots, a_i, \underbrace{\dots, a_j, \dots, a_k, \dots, a_n}_{\mathcal{A}_i = \{a_j, a_n\}}$$

- ▶  $a_i$  detects its local symmetries, and keeps them in  $p_i$  (but omits the ones involving agents  $\notin \{a_i\} \cup \mathcal{A}_i$ ).



## Proposition of algorithm (2/2)

- ▶  $a_i$  sends  $(\mathcal{A}_i, p_i)$  to the first agent in  $\mathcal{A}_i$

## Proposition of algorithm (2/2)

- ▶  $a_i$  sends  $(\mathcal{A}_i, p_i)$  to the first agent in  $\mathcal{A}_i$
- ▶ When  $a_j$  receives  $p_i$ , it builds  $p_{i,j}$ , by taking out the permutations which are not consistent with  $a_j$ . Then,



## Proposition of algorithm (2/2)

- ▶  $a_i$  sends  $(\mathcal{A}_i, p_i)$  to the first agent in  $\mathcal{A}_i$
- ▶ When  $a_j$  receives  $p_i$ , it builds  $p_{i,j}$ , by taking out the permutations which are not consistent with  $a_j$ . Then,
  - ▶ otherwise, we send  $p_{i,j}$  to the next agent in  $\alpha_i$ .

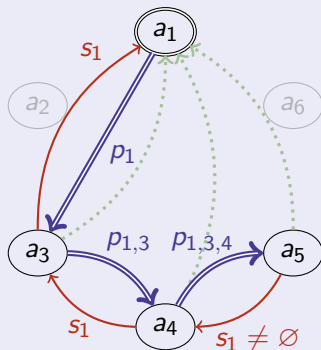
## Proposition of algorithm (2/2)

- ▶  $a_i$  sends  $(\mathcal{A}_i, p_i)$  to the first agent in  $\mathcal{A}_i$
- ▶ When  $a_j$  receives  $p_i$ , it builds  $p_{i,j}$ , by taking out the permutations which are not consistent with  $a_j$ . Then,
  - ▶ if  $p_{i,j} = \emptyset$ , we stop the process and **send a no to the agent  $a_i$**  who initiated the process;
  - ▶ otherwise, we **send  $p_{i,j}$  to the next agent in  $\alpha_i$** .

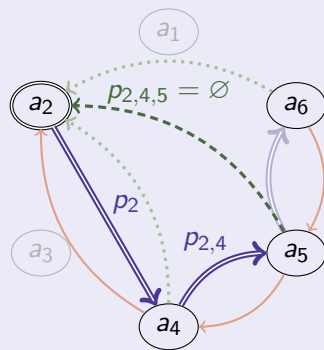
## Proposition of algorithm (2/2)

- ▶  $a_i$  sends  $(\mathcal{A}_i, p_i)$  to the first agent in  $\mathcal{A}_i$
- ▶ When  $a_j$  receives  $p_i$ , it builds  $p_{i,j}$ , by taking out the permutations which are not consistent with  $a_j$ . Then,
  - ▶ if  $p_{i,j} = \emptyset$ , we stop the process and send a no to the agent  $a_i$  who initiated the process;
  - ▶ if  $a_j$  is the last agent in  $\mathcal{A}_i$ , we send  $p_{i,j}$  back;
  - ▶ otherwise, we send  $p_{i,j}$  to the next agent in  $\alpha_i$ .

## Message flow in the detection/validation



$$\mathcal{A}_1 = \{a_3, a_4, a_5\}$$



$$\mathcal{A}_2 = \{a_4, a_5, a_6\}$$

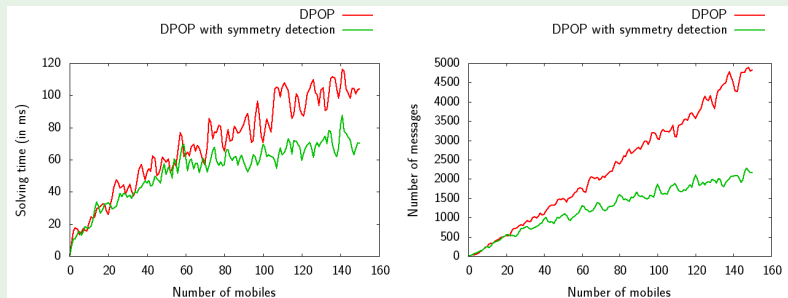
## Termination

- ▶ Each agent starts one process, ending with the reception of either a no or a symmetry message.

## Complexity

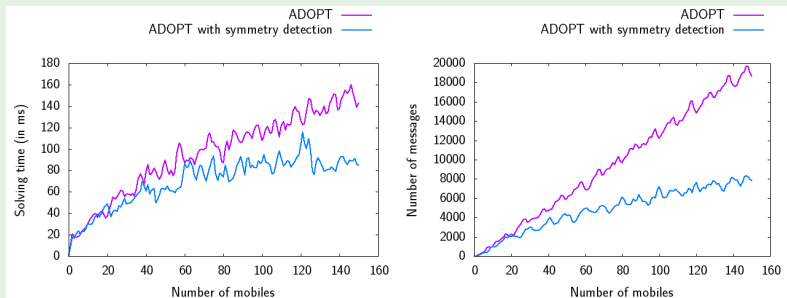
- ▶  $m_{preprocessing} = O(n \cdot k) \ll \Delta m_{communication}$ , for  $n$  agents having  $k = \max |\mathcal{A}_i|$  neighbours at most.

## Performance for 25 sensors with DPOP



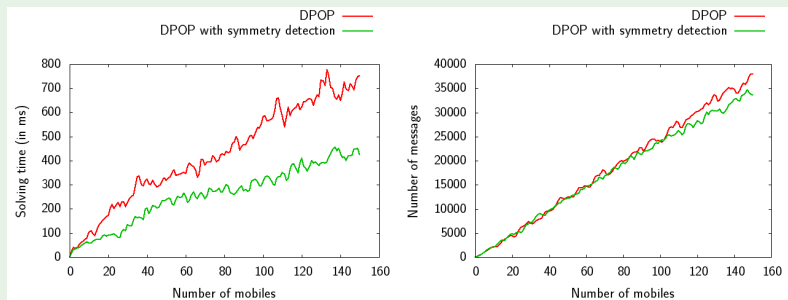
- ▶ Solving time (left) and number of messages (right)
- ▶ Solving time improved by 30% and number of messages cut down by 50%.

## Performance for 25 sensors with ADOPT



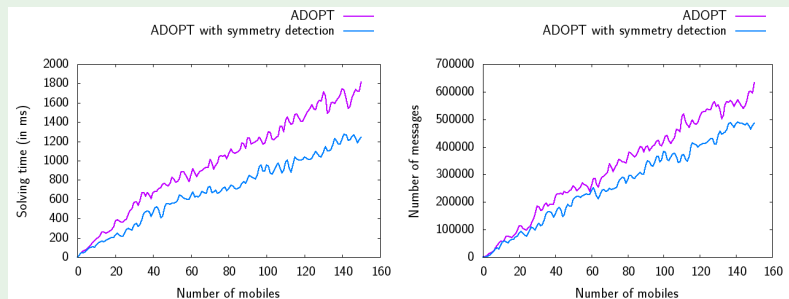
- ▶ Solving time (left) and number of messages (right)
- ▶ Solving time improved by 40% and number of messages cut down by 50%.

## Performance for 81 sensors with DPOP



- ▶ Solving time (left) and number of messages (right)
- ▶ Solving time improved by 40%.

## Performance for 81 sensors with ADOPT



- ▶ Solving time (left) and number of messages (right)
- ▶ Solving time improved by 30% and number of messages cut down by 20%.

## Conclusion

---

- ▶ Basic preprocessing method for dealing with symmetries in a distributed context.
- ▶ Significant improvements on several `SENSORDCSP` instances.
- ▶ Validate on other problems. . .
- ▶ Merge (better) local search and distributed process?
- ▶ Merge symmetry detection with DFS construction?