Distributed Forward Checking with Conflict Based Backjumping and Dynamic Ordering

Amnon Meisels and Igor Razgon

Department of Computer Science
Ben-Gurion University of the Negev
Beer-Sheva, 84-105, Israel
{am,irazgon}@cs.bgu.ac.il

Abstract. Dinamically ordered distributed forward checking (DODFC) [6] is a distributed constraint solver having a high level of cooperation. In this paper we introduce an extension of DODFC called DODFC-CBJ wich is a distributed analogue of a well-known FC-CBJ algorithm [8]. Correctness and completeness proof of DODFC-CBJ are outlined. It is proved that, applied on the same input, DODFC-CBJ performs no more consistency checks and sends no more messages than DODFC. Preliminary experimental results and further developments of the proposed algorithm are discussed.

1 Introduction

Distributed constraints networks (DisCNs) are composed of a set of agents, where each agent contains its own constraints network.

Agents are connected by constraints between some of their variables and variables of other agents. The problem of finding a globally consistent assignment of all variables of all agents is a distributed constraint satisfaction problem (DisCSP) (cf. [14]). Agents communicate by exchanging messages and distributed search algorithms for DisCSPs are run by all agents, each assigning values to its own variables, checking consistency of assignments with constraining agents. The first distributed search algorithms for DisCSPs were proposed by Yokoo [14–16]. In many ways the most successful distributed algorithm was the asynchronous backtrack (ABT) [16] and several modifications to ABT have been investigated in the last three years [1, 11].

Formally, a distributed constraints network (DisCN) is defined to be a set of agents $A_1, A_2, ..., A_k$. Each agent A_i contains a set of constrained variables $X_{i_1}, X_{i_2}, ..., X_{i_{n_i}}$, each variable X_{i_j} with its domain of values D_{i_j}. Agents are connected by constraints between variables that belong to different agents.

Constraints or **relations** R of a DisCN are subsets of the Cartesian product of the domains of the constrained variables.

For a set of constrained variables $X_{i_k}, X_{j_l}, ..., X_{m_n}$ the constraint is defined as $R \subseteq D_{i_k} \times D_{j_l} \times ... \times D_{m_n}$. with domains of values for each variable $D_{i_k}, D_{j_l}, ..., D_{m_n}$. As an example, a **binary constraint** R_{ij} between any two variables in the DisCN X_j and X_i is a subset of the Cartesian product of their domains; $R_{ij} \subseteq D_j \times D_i$.

The case where $i = j$ denotes a **unary constraint** on X_i. An assignment (or a label) of a DisCN is a pair $< var, val >$, where *var* is a variable of the network and *val* is a value from *var*'s domain that is assigned to it. A *partial solution* is a set of assignments (for some set of variables).

A **solution** P to a DisCN is a tuple (i.e. assignments of values to all variables) that satisfies all the constraints.

Dynamically ordered distributed forward checking algorithm (DODFC) was proposed in [6]. Contrary to majority of distributed constraint solvers performing assignments of variables asynchronously and independently, execution of DODFC is based on strong coordination between agents. At all times of its execution, the current state of DODFC contains only one partial solution which is consistent (not violating any constraint), while current states of asynchronous solvers may contain a number of possibly inconsistent partial solutions.

The assignment process is based on cooperation of all agents in the network. The Main Agent (MA), holding the variable to be assigned (*the current variable*) collects from other agents all necessary data concerning estimation of values of the current variable. Then the MA assigns the current variable with a value having the highest estimation and selects the next current variable. The next MA is the agent holding the new current variable. If the MA can not assign the current variable, it sends a backtrack message to the agent that was the MA in the previous iteration. The agent receiving a backtrack message initiates backtrack.

Experimental results show that DODFC is compatible with recent distributed constraint solvers. Moreover, it has a high level of cooperation. The number of constraint checks performed by DODFC approximately $n - 1$ times less than the number of constraint checks performed by its centralized version, where n is the number of agents.

In this paper we present an extension of DODFC called DODFC-CBJ, which is a distributed analogue of FC-CBJ [8]. This paper is organized as follows. In Section 2 we recall DODFC, in Section 3 we present DODFC-CBJ, outline its correctness and completeness proof, and prove that when DODFC and DODFC-CBJ are applied one the same input, DODFC-CBJ performs no more consistency checks (CC) and sends no more messages than DODFC. In Section 4 we provide preliminary experimental results and discuss further developments of the proposed algorithm.

2 The DODFC algorithm

In this section we recall the DODFC algorithm. The centralized version of DODFC is forward checking with variable and value ordering. Variables are ordered with the fail-first heuristic [5], the values of the current domain of the current variable are ordered in the decreased order of their compatibility. The notion of compatibility is defined below.

Definition 1. *Compatibility of assignment* $\langle V, val \rangle$ *with variable* V' *is the percent of values of the current domain of* V' *compatible with* $\langle V, val \rangle$ *.The compatibility of assignment with a set of variables is the minimum compatibility of this assignment with a variable from this set.*

A pseudocode of DODFC is given in Algorithm 1, 2, 3, 4. The following list enumerates our assumptions with regard to this algorithm.

1. The processed constraint network contains binary constraints only.
2. All agents except one begin code execution from the block REGULAR. Let's call these agents Regular Agents (RA).
3. The initial Main Agent (MA) has to be selected before invocation of the algorithm. This agent begins its execution from the block DECISION.
4. All agents are "aware" who is the initial MA.
5. All variables are initially unassigned.
6. The variable *step_number* refers to the number of the current assignment (or to the size of the current partial solution). Initially *step_number* is equal 0.

It is possible to represent execution of DODFC as a sequence of actions each of them may be either assignment, or backtrack, or solution, or fail. Let us follow the process of assignment execution (this process includes possibility of solution return).

1. Every RA sends to the MA the minimal domain size of its local unassigned variable (line 1 of Algorithm 1). Then it begins the cycle of reading messages and replying them (lines 4-30 of Algorithm 1).
2. The MA collects minimal domain sizes of all agents having unassigned variables (including information about its own local variables) (line 1, Algorithm 2).
3. The MA selects the agent that sent the minimal domain size. This agent becomes the MA in the current iteration. If this new MA is the current MA itself, it begins execution of the NEW_ASSIGNMENT block (line 3, Algorithm 2). Otherwise it sends to the selected RA a message informing the receiver that it was selected to be the MA in the current iteration, and becomes a RA (line 6, Algorithm 2). The selected RA receives this message and begins to execute the NEW_ASSIGNMENT block (lines 16-18 of Algorithm 1).
4. The MA sends estimation requests for all values of the current domain of the variable it tries to assign (line 1, Algorithm 3).
5. An RA obtaining this estimation request answers it (lines 9-11 of Algorithm 1). That is, for all values in the request, it computes their compatibility with its local variables.

Algorithm 1 DODFC:REGULAR

```
if there are unassigned variables then
    Send to the MA the minimal domain size among all unassigned variables
end if
while TRUE do
    Read a message msg
    if msg is a FAIL message then
        Return FAIL
    end if
    if msg is an estimation request then
        For each value in the request, compute its compatibility with my local unassigned variables and return answers to the sender
    end if
    if msg is a backtrack message then
        Rollback to the msg.step_number and update domains
        Run OLD_ASSIGNMENT
    end if
    if msg informs the selection to be the MA then
        Run NEW_ASSIGNMENT
    end if
    if msg is a refinement message then
        if msg.step_number < step_number then
            Rollback to the step msg.step_number and update domains
        end if
        Refine domain of unassigned variables
        step_number = step_number + 1
        Run REGULAR
    end if
    if msg is the solution message then
        Return solution
    end if
end while
```

Algorithm 2 DODFC:DECISION

```
Collect all necessary domain measures (including measures of my local variables) and determine the next MA
if This is the MA then
    Run NEW_ASSIGNMENT
else
    Send a selection message to the MA
    Run REGULAR
end if
```

Algorithm 3 DODFC:NEW_ASSIGNMENT

```
Send to all agents estimation requests with regard to all values of the current
domain of the assigned variable.
Receive all required answers (including the compatibility measures computed by
myself)
Compute estimations for all values (for every value select the minimum of its esti-
mations)
Sort values in the decreasing order of their estimations
Eliminate all values with zero estimations
Run OLD_ASSIGNMENT
```

Algorithm 4 DODFC:OLD_ASSIGNMENT

```
if The domain of the assigned variable is not empty then
  Assign the current variable with the first value from its domain and eliminate
  the assigned value from the current domain of the variable
  Refine the domains of my local unassigned variables
  step_number = step_number + 1
  if step_number is equal to the number of variables then
    Send to all agents the SOLUTION message
    Return solution
  else
    Send refinement messages with the assigned value to all agents having unas-
    signed variables
    Run DECISION
  end if
else
  if step_number = 0 then
    Send fail messages to all agents
    Return FAIL
  else
    if this was the MA in the previous step then
      Rollback to the previous step (decrease step_number by 1 and update do-
      mains)
      Run OLD_ASSIGNMENT
    else
      Send a backtrack message with the value step_number − 1 to the agent that
      was the MA in the previous iteration
      Run REGULAR
    end if
  end if
end if
```

6. The MA obtains all necessary replies (including computed by the MA itself for its local variables) (line 2 of Algorithm 3).
7. Based on the obtained assessments, the MA computes estimations for all values of its current domain. (The estimation for each value is the minimum of the obtained set of estimations of this value, by our definition of compatibility). Then the MA sorts all values in the decreasing order of their estimations, eliminates from the current domain all values having zero estimation and begins to execute the OLD_ASSIGNMENT block (lines 4-6 of Algorithm 3).
8. To proceed without backtrack, the MA should have ant least one value in its current domain. This condition is checked in line 1 of Algorithm 4. Then the MA makes assignment, eliminates the assigned value from the current domain of the current variable, refines domains of its own local variables, and increases the *step_number* value (lines 2-4 of Algorithm 4).
9. At this point, the MA has two options. If it detects that all variables are assigned already (*step_number* is equal to the number of variables), it initiates the process of solution return (lines 5-8 of Algorithm 4, lines 26-28 of Algorithm 1). Otheriwse, it initiates lookahead process and begins to execute the DECISION block (lines 8-10 of Algorithm 4).
10. When an RA receives a lookahead message, it eliminates from the current domains of its unassigned variables all values that are incompatible with the received assignment (lines 19-26 of Algorithm 1). This part of pseudocode contains a subtle point (lines 20-22): the RA replaces its *step_number* variable by the value of *step_number* of the MA. As we will see below, this coordination is needed for backtrack processing. Then the RA passes to the next iteration.

Backtrack is performed when the MA detects that its current domain is empty (line 1, Algorithm 4). When agents execute backtrack, they rollback to the state that was just before the last assignment and try to reassign the variable that was assigned last.

Backtrack (and fail) processing of DODFC contains the following stages:

1. If the current compound label is empty, the MA initiates fail processing (lines 13-16 of Algorithm 4). It sends fail messages to all RA's and fails. When an RA receives its fail message, it also fails (lines 6-8 of Algorithm 1).
2. If there are assigned variables, the MA holding the last assigned variable is informed about backtrack and performs rollbacks to the previous iteration. The new MA initiates the backtrack process by sending to all RA's refinement messages with the new value of *step_number*. Now it is clear why does an RA performing assignment need replace its *step_number* value by the *step_number* value obtained from the refinement message. If the obtained value is less than its own, this means that the MA initiates backtrack and the RA has to rollback to the step indicated in the message.
3. A number of consecutive backtracks is possible if the current domains of one or more "recently" assigned variables are empty. In this case the backtrack

process described above repeats. This fact can cause that in lines 12-15 and 20-22 of Algorithm 1 *msg.step_number* is less than *step_number* by more than 1.

Note that majority of DODFC computational procedures, such as computation of minimal domain sizes, estimation of domain values of the current variable, elimination of values incompatible with the assignment of the current variable are performed in parallel by all agents. That is, DODFC has a high level of cooperation. By the way, parallel work of agents in other distributed constraint solvers [17, 11] is not cooperative, because every agent in these solvers computes its assignments independently.

3 From DODFC to DODFC-CBJ

3.1 Description of the Algorithm

To understand the extension of DODFC to DODFC-CBJ consider again the process of backtrack of DODFC.

- The current MA detects that the current domain of the current variable is empty and informs about this fact the agent that was the MA in the previous iteration.
- The new MA restores the current state of the previous iteration and if the current domain of the current variable is not empty, it sends to all agents refinement messages with the new value of *step_number*.
- An RA receiving the refinement message detects that the *step_number* value of the message is less than its own value of *step_number*. Therefore it replaces its *step_number* by the value from the message and restores the current state corresponding to the new value of *step_number*.

This backtrack scheme well suites for distributed backjumping, because it allows to backtrack to an arbitrary number of steps. For example, when an RA gets a refinement message, the *step_number* value of this message must not be its own *step_number* value minus one. In essence, such cases occur even for DODFC when there is a number of consecutive backtracks.

Therefore, the only subtle point concerning extension of DODFC to DODFC-CBJ is maintenance of *conflict sets* [8]. Conflict set of an unassigned variable V is a subset of the current partial solution, which is "responsible" for elimination of values from the current domain of V. Let us consider the mechanism of construction of conflict sets for FC-CBJ.

Initially conflict sets of all variables are empty. The conflict set of variable V is updated when some value is eliminated from the current domain of V. For both FC and FC-CBJ there are three possible ways of elimination of values.

FC-refinement: A domain value of an unassigned variable V is eliminated from the current domain of V because it is incompatible with the assignment of the current variable.

Unsuccessful assignment: A value of the current domain of the current variable is eliminated because it is incompatible with all values of the current domain of some unassigned variable V.

Backtrack: The current variable can't be assigned, because all values were eliminated from its current domain. In this case, the algorithm changes the current partial solution.

In the case of FC-refinement the assignment of the current variable is inserted in the conflict set of the variable whose value is eliminated. In the second case, the conflict set of V is added to the conflict set of the current variable. The case of backtrack is a little more complicated. In consists of the following steps:

1. The variable V', the algorithm backtracks to is determined. It is the variable whose assignment is last (chronologically) in the conflict set of the current variable.
2. The assignment of V' is eliminated from the conflict set of V'.
3. The obtained set is added to the conflict set of V' (of course, with elimination of the current assignment of V').

To implement construction of conflict sets for DODFC-CBJ, agents should be able to inform one another about their conflict sets. In particular, formats of backtrack and estimation messages should be changed as follows:

- The MA sending a backtrack message to an RA (line 21 of Algorithm 8) includes in this message the conflict set of the current variable.
- An RA, sending a zero estimation to the MA, adds to the estimation the conflict set of its local unassigned variable, all current domain values of which are incompatible with the zero estimated value.

The pseudocode of DODFC-CBJ is shown in Algorithms 5, 6, 7, 8. All assumptions for DODFC hold also for DODFC-CBJ. The additional assumption is that initially conflict sets of all variables are empty

3.2 Analysis

Our analysis of DODFC-CBJ contains the following stages:

1. We define so-called stable states for DODFC and DODFC-CBJ.
2. We prove that the sequence of stable states generated by DODFC-CBJ on a particular input is a subsequence of the sequence generated by DODFC on the same input. Moreover, for every pair of corresponding states of these sequences, the current partial solution, the current variable and the current domains of all variables are the same (up to random breaking of ties).
3. Given correctness and completeness of DODFC proved [6], we deduce that
   - DODFC-CBJ is correct and complete.
   - DODFC-CBJ performs no more consistency checks and sends no more messages than DODFC applied on the same input.

Algorithm 5 DODFC-CBJ: REGULAR

```
if there are local unassigned variables then
  Send to the MA the minimal domain size among all unassigned variables
end if
while TRUE do
  Read a message msg
  if msg is a FAIL message then
    Return FAIL
  end if
  if msg is an estimation request then
    For each value in the request, compute its compatibility with my local unas-
    signed variables
    Return answers to the sender accompanying every zero estimation with the
    conflict set of a local variable incompatible with the zero-estimated value
  end if
  if msg is a backtrack message then
    Rollback to the msg.new_step_number and update domains
    Add msg.add_conf_set to the conflict of the current variable
    Run OLD_ASSIGNMENT
  end if
  if msg informs the selection to be the MA then
    Run NEW_ASSIGNMENT
  end if
  if msg is a refinement message then
    if msg.step_number < step_number then
      Rollback to the step msg.step_number and update domains
    end if
    Refine domain of unassigned variables
    If the refinement led to elimination of some values from current domains of
    unassigned variables, add the assignment of the current variable to the conflict
    sets of these values
    step_number = step_number + 1
    Run REGULAR
  end if
  if msg is the solution message then
    Return solution
  end if
end while
```

Algorithm 6 DODFC-CBJ:DECISION

```
1: Collect all necessary domain measures (including measures of my local variables)
   and determine the next MA
2: if This is the MA then
3:    Run NEW_ASSIGNMENT
4: else
5:    Send a selection message to the MA
6:    Run REGULAR
7: end if
```

Algorithm 7 DODFC-CBJ:NEW_ASSIGNMENT

```
1: Send to all agents estimation requests with regard to all values of the current
   domain of the assigned variable.
2: Receive all required answers (including compatibility measures computed by my-
   self)
3: Compute estimations for all values (for every value select the minimum of its esti-
   mations)
4: Sort values in the decreasing order of their estimations
5: Eliminate all values with zero estimations
6: For every eliminated value, add a conflict set accompanying a zero estimation of
   this value to the conflict set of the current variable
7: Run OLD_ASSIGNMENT
```

Algorithm 8 DODFC-CBJ:OLD_ASSIGNMENT

```
if the domain of the assigned variable is not empty then
    Assign the current variable with the first value of its domain and eliminate the assigned value from the current domain of the variable
    Refine the domains of my local unassigned variables
    step_number = step_number + 1
    if step_number is equal to the number of variables then
        Send to all agents the SOLUTION message
        Return solution
    else
        Send refinement messages with the assigned value to all agents having unassigned variables and refine my own unassigned variables
        Run DECISION
    end if
else
    if step_number = 0 then
        Send fail messages to all agents
        Return FAIL
    else
        Compute new_step_number, the number of assignment backjump is executed to
        Compute the set add_conf_set that should be added to the conflict set of the new current variable
        if this was the MA in the step new_step_number then
            Rollback to the step new_step_number (replace step_number by new_step_number, update domains, and add add_conf_set to the conflict set of the new current variable)
            Run OLD_ASSIGNMENT
        else
            Send a backtrack message with the value new_step_number and add_conf_set to the agent that was the MA in the step new_step_number
            Run REGULAR
        end if
    end if
end if
```

Let us consider the following states that may occur during execution of both DODFC and DODFC-CBJ:

SELECTION: All minimal domain values are already computed and the new current variable has to be selected.
ASSIGNMENT: Estimations for all values of the current domain of the current variable are known. Now the MA should either assign the current variable with a value having the maximal estimation or backtrack if the current domain of the current variable is empty.
SOLUTION: Solution is returned.
FAIL: The algorithm deduced that the given constraint network has no solution.

Everyone of these states is "stable" in the sense that for every state we exactly know what is every agent doing. For example, for the *selection* state, we know that the MA is preparing to select the new current variable and all RA's are waiting for a message from the MA.

Both DODFC and DODFC-CBJ generate sequence of these states during their execution. The following lemma can be formulated.

Lemma 1. *The sequence of states generated by DODFC-CBJ on some input is a subsequence of the sequence of states generated by DODFC on the same input. Moreover, for every pair of corresponding states of these sequences, the current partial solution, the current variable and the current domains of all variables are the same (up to random breaking of ties).*

Outline of Proof.

Let Sq be the sequence generated by DODFC-CBJ on some input. Let Sq' be the sequence generated by DODFC on the same input. Let $Sq[1..i]$ and $Sq'[1..i]$ be the prefixes of length i of Sq and Sq' respectively. Let also Sq_i and Sq'_i be the i-th states of Sq and Sq' respectively.

Now we can reformulate Lemma 1 as follows:

Any prefix of Sq is a subsequence of Sq'. Moreover, for every pair of corresponding states their CSP data are the same.

Let us prove this lemma by induction of the length of the prefix of Sq. This lemma trivially holds for $Sq[1..1]$, because in both algorithms RA's start from sending to the MA minimal domain sizes of their local variables (we assume that the first MA for both algorithms is the same).

Assume that this lemma holds for $Sq[1..i]$. Let us prove its validity for $Sq[1..i+1]$. Let Sq'_k be the state of Sq' corresponding to Sq_i. Existence of such state follows from the induction assumption. If, passing from Sq_i to Sq_{i+1}, DODFC-CBJ doesn't perform backjump of length more than one, then $Sq_{i+1} = Sq'_{k+1}$, their current partial solutions, current variables and current domains are the same, and this lemma holds for $Sq[1..i+1]$.

Otherwise, both DODFC-CBJ and DODFC perform backtrack. Therefore $Sq_{i+1} = Sq'_{k+1}$, but their CSP data may be different. Given proved that transition from Sq'_k to Sq'_{k+1} is successful, the transition from Sq_i to Sq_{i+1} is also successful. It remains to prove that DODFC eventually arrives to the state corresponding to Sq_{i+1}. To show this we have to prove that, passing to Sq_{i+1}, DODFC-CBJ doesn't loss solutions. This fact may be proved by the following lemma:

Lemma 2. *Let P ,V be the current partial solution and an unassigned variable at some moment of execution of DODFC-CBJ. Let also val be a value eliminated from the current domain of V. Then the extension of P by $\langle V, val\rangle$ creates a nogood (a partial solution that can not be a part of any complete solution).*

We omit here the proof of this lemma. It is quite technical. We order chronologically values eliminated ever from domains of their variables and prove this lemma by induction for every element of the obtained sequence.

Therefore, based on correctness and completeness of DODFC and the fact that DODFC can't execute backtracks of length more than 1, me deduce that DODFC eventually arrives to the state corresponding to Sq_{i+1}, that is Lemma 1 holds for $Sq[1..i+1]$, which proves the lemma. $\square$

Now let us prove several corollaries of Lemma 1.

Corollary 1. *DODFC-CBJ terminates.*

Immediately follows from Lemma 1 and termination of DODFC.

Corollary 2. *If DODFC-CBJ and DODFC are applied on the same input, their final states are the same.*

Proof According to Corollary 1 DODFC-CBJ terminates. The final state of DODFC-CBJ may be either *solution* or *fail*. According to Lemma 1, this state belongs to the sequence of states generated by DODFC applied on the same input. But such state may occur for DODFC as the final state only. $\square$

Corollary 3. *DODFC-CBJ is correct and complete.*

Immediately follows from correctness and completeness of DODFC and Corollaries 1 and 2.

Corollary 4. *When DODFC and DODFC-CBJ are applied on the same input, the number of CC's performed by DODFC-CBJ doesn't exceed the number of CC's performed by DODFC, and the number of messages sent by DODFC-CBJ doesn't exceed the number of messages sent by DODFC.*

Proof.
Let Sq and Sq' are the sequences generated by DODFC and DODFC-CBJ respectively when they applied on the same input. Then, based on Lemma 1, we can prove the following claim, stronger than Corollary 4.

Let Sq_i and Sq_{i+1} be two consecutive states of Sq. Let also Sq'_k and Sq'_l be states of Sq' corresponding to Sq_i and Sq_{i+1} respectively. Then, passing from Sq_i to Sq_{i+1}, DODFC-CBJ performs no more constraints checks and send no more messages than DODFC passing from Sq'_k to Sq'_l.

Really if DODFC-CBJ doesn't executes backjump of length more than one when passing from Sq_i to Sq_{i+1} then the numbers of constraint checks performed and the number of messages sent are the same for both algorithms. Otherwise DODFC performs more constraint checks and sends more messages than DODFC-CBJ. [1]

4 Discussion

In this paper we presented a DODFC-CBJ algorithm which extends DODFC [6]. The proposed algorithm is a distributed version of a well- known FC-CBJ algorithm of Prosser [8]. We outlined correctness and completeness proof of DODFC-CBJ and proved that DODFC-CBJ performs not worse than DODFC on any input (actually, it frequently outperforms DODFC). By the way, our proof can be applied to prove correctness and completeness of FC-CBJ and its comparison with FC.

Preliminary experiments show that DODFC-CBJ like DODFC has a high level of cooperation. In particular, if every agent holds more than one variable, than the number of consistency checks performed by DODFC-CBJ is $n-1$ times less than than the number of consistency checks performed by FC-CBJ. [2].

To conclude, let us present a number of additional optimizations that may improve performance of both DODFC-CBJ and FC-CBJ.

1. Consider again the process of conflict set construction during unsuccessful assignment. We add to the conflict set of the current variable the conflict set of an unassigned variable that led to zero estimation of the unsuccessful assignment. It is possible that the unsuccessful assignment got more than one zero estimations from different agents. Then the MA can select the conflict set to append to its own conflict set. Various selection heuristics may be applied here. For example, select the conflict set with the minimal last assignment.
2. Conflict sets of variables may be split into conflict sets of values eliminated from current domains of their variables. Then every eliminated value has its own conflict set explaining its elimination (eliminating explanation in terms of Ginsberg [3]). Based on the splitting the following optimizations are possible:
   – The process of backtrack may be modified. In particular, when agents rollback to a previous state, they may restore in their current domains only those values whose conflict sets are irrelevant (not a part of the current partial solution). Note that this facility is presented in Dynamic Backtracking.

[1] Sq'_k and Sq'_l are not consecutive, and even transition to Sq'_{k+1} takes the same number of constrain checks and messages as transition of DODFC-CBJ to Sq_{i+1}

[2] We take FC-CBJ with the same variable and value ordering heuristics as DODFC-CBJ

– We can optimize the length of constructed conflict sets. For example, let V be the current variable. Assume that its value val is eliminated because $\langle V, val \rangle$ is incompatible with all values of the current domain of some unassigned variable V'. Than, instead of setting the conflict set of val to be the union of conflict sets of all values eliminated from V', we set it to be the union of those values of V' that are *not in conflict* with $\langle V, val \rangle$. A similar optimization is possible for the case of backtrack.

References

1. C. Bessire, A. Maestre, and P. Meseguer. *Distributed dynamic backtracking* Proceedings of the IJCAI'01 workshop on Distributed Constraint Reasoning, Seattle WA, pages 9-16.
2. D. Frost, R. Dechter, *Look-ahead value ordering for constraint satisfaction problems*, Proceedings of IJCAI-95, pp. 572-578, Montreal, Canada, August 1995.
3. M. L. Ginsberg, *Dynamic Backtracking*,Journal of Artificial Intelligence Research, volume 1, pp. 25-46, 1993.
4. Y. Hamadi, C. Bessiere *Bakctracking in Distributed Constraint Networks*, Proc. ECAI-98, pp. 219-223, Brighton, August 1998.
5. R.M. Haralick, G.L.Eliott *Increasing tree search efficiency for constraint satisfaction problems*
6. A. Meisels, I. Razgon *Distributed Forward Checking with Dynamic Ordering*, in Proceedings of CoSolv-01 Workshop, CP2001, Paphos, Cyprus, 2001
7. P. Meseguer M. A. Jimenez *Distributed Forward Checking*, Proc. CP-2000 Workshop on Distributed Constraint Satisfaction, Singapore, 22 September, 2000.
8. P. Prosser, *Hybrid Algorithms for the Constraint Satisfaction Problem* , Computational Intelligence, volume 9, pp. 268-299, 1993.
9. P. Prosser *Binary constraint satisfaction problems: some are harder than others* , Proc. ECAI-94, pp.95-99, 1994.
10. M. C. Silaghi, D. Sam-Haroud, B. Faltings, *Asynchronous Search With Aggregations*, AAAI/IAAI2000, pp. 917-922, August 2000.
11. M.C.Silaghi,D.Sam-Haroud,B.Faltings, *ABT with Asynchronous Reordering*, 2nd Asia-Pacific IAT,Maebashi,Japan,2001.
12. G. Solotorevsky, E. Gudes, A. Meisels *Modeling and Solving Distributed Constraint Satisfaction Problems (DCSPs).*, Constraint Processing-96, New Hamphshire, October 1996.
13. E.Tsang *Foundations of Constraint Satisfaction*, Academic Press Limited, 1993.
14. M. Yokoo et. al. *Distributed Constraint Satisfaction for Formalizing Distributed Problem Solving*, IEEE Intern. Conf. Distrb. Comp. Sys., pp. 614-621, 1992.
15. M. Yokoo *Asynchronous Weak-commitment Search for solving Distributed Constraint Satisfaction Problems*, Proc. CP95, pp. 88-102, Cassis, France, 1995.
16. M. Yokoo, E. H. Durfee, T. Ishida, K. Kuwabara *Distributed Constraint Satisfaction Probelm: Formalization and Algorithms*, IEEE Trans. on Data and Kn. Eng., 10 (5), pp. 673-685, 1998.
17. M.Yokoo, K.Hirayama *Distributed Constraint Satisfaction Algorithm for Complex Local Problems*, Inter. Conf. Multi Agent Systems, pp. 372-379, La Villette, Paris, July, 1998.