Asynchronous Forward-Bounding for Distributed Constraints Optimization

Amir Gershman and Amnon Meisels and Roie Zivan

Abstract.
A new search algorithm for solving distributed constraint optimization problems (DisCOPs) is presented. Agents assign variables sequentially and propagate their assignments asynchronously. The asynchronous forward-bounding algorithm (AFB) is a distributed optimization search algorithm that keeps one consistent partial assignment at all times. Forward bounding propagates the bounds on the cost of solutions by sending copies of the partial assignment to all unassigned agents concurrently. The algorithm is described in detail and its correctness proven. Experimental evaluation of AFB on random Max-DisCSPs reveals a phase transition as the tightness of the problem increases. This effect is analogous to the phase transition of Max-CSP when local consistency maintenance is applied [3]. AFB outperforms Synchronous Branch & Bound (SBB) as well as the asynchronous state-of-the-art ADOPT algorithm, for the harder problem instances. Both asynchronous algorithms outperform SBB by a large factor.

1 Introduction
The Distributed Constraint Optimization Problem (DisCOP) is a general framework for distributed problem solving that has a wide range of applications in Multi-Agent Systems and has generated significant interest from researchers [12, 18]. Distributed constraints optimization problems (DisCOPs) are composed of agents, each holding its local constraints network, that are connected by constraints among variables of different agents. Agents assign values to variables and communicate with each other, attempting to generate a solution that is globally optimal with respect to the costs of constraints between agents (cf. [12, 14]).

DisCOPs are an elegant model for many everyday combinatorial problems that are distributed by nature. Take for example a large hospital that is composed of many wards. Each ward constructs a weekly timetable assigning its nurses to shifts. The construction of a weekly timetable involves solving a constraint optimization problem for each ward. Some of the nurses in every ward are qualified to work in the Emergency Room. Hospital regulations require a certain number of qualified nurses (e.g. for Emergency Room) in each shift. This imposes constraints among the timetables of different wards and generates a complex Distributed COP [16, 2].

Several distributed search algorithms for DisCOPs have been proposed in recent years [17, 12, 14]. The present paper proposes a new distributed search algorithm for DisCOPs, Asynchronous Forward-Bounding (AFB). In the AFB algorithm agents assign their variables sequentially to generate a partial solution. The innovation of the proposed algorithm lies in propagating partial solutions asynchronously. Propagation of partial solutions enables asynchronous updating of bounds on their cost, and early detection of a need to backtrack, hence the algorithm’s name - Asynchronous Forward Bounding.

The overall framework of the AFB algorithm is based on a Branch and Bound scheme. Agents extend a partial solution as long as the lower bound on its cost does not exceed the global bound, which is the cost of the best solution found so far. In the proposed AFB algorithm, (similar to the AFC algorithm for DisCSPs [11]) the state of the search process is represented by a data structure called Current Partial Assignment (CPA). The CPA starts empty at some initializing agent that records its assignments on it and sends it to the next agent. The cost of a CPA is the accumulated cost of constraints involving the value assignments it contains. Each agent which receives the CPA, adds assignments of its local variables such that the CPA’s cost will not exceed the global upper bound. If it cannot find such assignments, it backtracks by sending the CPA to the last assigning agent, requesting it to revise its assignment.

The innovation of the AFB algorithm lies in the asynchronous use of the CPAs by the participating agents. An agent that succeeds to extend the assignment on the CPA sends forward copies of the updated CPA, requesting all unassigned agents to compute lower bound estimations on the cost of the partial assignment (PA). The assigning agent will receive these estimations asynchronously over time and use them to update the lower bound of the CPA. Using these bounds, the assigning agent can detect if any extension of this partial assignment into a complete assignment will cause it to exceed the global upper bound, and in such cases it will backtrack. Backtracking is performed by creating a new CPA, a clone of the previous one, and sending it to the last assigning agent, to have that assignment replaced. A time stamp mechanism for forward checking in DisCSPs is used by agents to determine the most updated CPA and to discard obsolete CPAs (cf. [13]).

The concurrency of the AFB algorithm is achieved by the fact that forward-bounding is performed concurrently and asynchronously by all agents.

The AFB algorithm is described in detail in Section 3 and its correctness is proven in Section 4. The performance of AFB is compared to that of Synchronous Branch and Bound (SBB) and ADOPT on randomly generated DisCOPs. AFB outperforms SBB as well as the best asynchronous optimization algorithm ADOPT by a large factor, on the harder instances of random problems. This is true for two measures of distributed performance: the number of non concurrent steps of computation and the total number of messages sent (see section 5).

A phase transition for problems of very high tightness is observed for AFB, similarly to the one that is reported for extended local consistency maintenance on Max-CSPs [3, 4].

---

1 The research was supported by the Lynn and William Frankel Center for Computer Science, and by the Paul Ivanier Center for Robotics.

2 Department of Computer Science, Ben-Gurion University of the Negev, Beer-Sheva, 84-105, Israel. email: {amirger,am,zivanr}@cs.bgu.ac.il
2 Distributed Constraint Optimization

A distributed constraint optimization problem (DisCOP) is composed of a set of agents \( A_1, \ldots, A_n \), each holding a set of constrained variables. Each variable \( X_i \) has a domain \( D_{i} \) - a set of possible values. Constraints (or relations) exist between variables. Each constraint involves some variables (possibly belonging to different agents), and defines a non-negative cost for every possible value combination of these variables. A binary constraint is a constraint involving only two variables. An assignment (or a label) is a pair including a variable, and a value from that variable’s domain. A partial assignment (PA) is a set of assignments, in which each variable appears at most once. The cost of a partial assignment is computed using all constraints that involve only variables that appear in the partial assignment. All these costs are accumulated, and the sum is denoted as the cost of the partial assignment. A full assignment is a partial assignment that includes all the variables. A solution to the DisCOP is a full assignment of minimal cost.

A widely used special case of DisCOPs is to use a cost of 1 unit for each broken (unsatisfied) constraint. This type of problem is termed Max-DisCSPs, in accordance with Max-CSPs for the centralized case [3, 6]. In this paper, we will assume that each agent is assigned a single variable, and use the term “agent” and “variable” interchangeably. We will assume that constraints are at most binary and the delay in delivering a message is finite. Furthermore, we assume a static final order on the agents, known to all agents participating in the search process. These assumptions are commonly used in DisCSP and DisCOP algorithms [17, 12]. For the experimental section we used Max-DisCSPs [3, 12]. Max-DisCSPs are a special class of DisCOPs in which all costs of binary constraints are exactly one.

3 Asynchronous Forward Bounding - AFB

In the AFB algorithm a single most up-to-date current partial assignment is passed among the agents. Agents assign their variables only when they hold the up-to-date CPA. The CPA is a unique message that is passed between agents, and carries the partial assignment that agents attempt to extend into a complete and optimal solution by assigning their variables on it. The CPA also carries the accumulated cost of constraints between all assignments it contains, as well as a unique time-stamp.

Only one agent performs an assignment on a CPA at a time. Copies of the CPA are sent forward and are concurrently processed by multiple agents. Each unassigned agent computes a lower bound on the cost of assigning a value to its variable, and sends this bound back to the agent which performed the assignment. The assigning agent uses these bounds to prune sub-spaces of the search-space which do not contain a full assignment with a cost lower than the best full assignment found so far.

More specifically, every agent that adds its assignment to the CPA sends forward copies of the CPA, in messages we term \( FB_{CPA} \), to all agents whose assignments are not yet on the CPA. An agent receiving an \( FB_{CPA} \) message computes a lower bound on the cost increment caused by adding an assignment to its variable. This estimated cost is sent back to the agent who sent the \( FB_{CPA} \) message via \( FB_{ESTIMATE} \) messages. The following two paragraphs describe how exactly this estimation is computed.

Denote by \( cost((i, v), (j, u)) \) the cost of assigning \( A_i = v \) and \( A_j = u \). For each agent \( A_i \) and each value in its domain \( v \in D_i \), we denote the minimal cost of the assignment \((i, v) \) incurred by an agent \( A_j \) by \( h_j(v) = min_{u \in D_j}(cost((i, v), (j, u))) \). We define \( h(v) \), the total cost of assigning the value \( v \), to be the sum of \( h_j(v) \) over all \( j > i \). Intuitively, \( h(v) \) is a lower bound on the cost of constraints involving the assignment \( A_i = v \) and all agents \( A_j \) such that \( j > i \). Note that this bound can be computed once per agent, since it is independent of the assignments of higher priority agents.

An agent \( A_i \), which receives an \( FB_{CPA} \) message, can compute for every \( v \in D_i \) both the cost increment of assigning \( v \) as its value, i.e. the sum of the cost of conflicts \( h \) has with the assignments included in the CPA, and \( h(v) \). The sum of these, is denoted by \( f(v) \). The lowest calculated \( f(v) \) among all values \( v \in D_i \) is chosen to be the lower bound estimation of agent \( A_i \).

Figure 1 presents a constraint network, in which \( A_1 \) already assigned the value \( v_1 \) and \( A_2, A_3, A_4 \) are unassigned. Let us assume that the cost of every constraint is one. The cost of \( v_3 \) will increase by one due to its constraint with the current assignment thus \( f(v_3) = 1 \). Since \( v_4 \) is constrained with both \( v_8 \) and \( v_9 \), assigning this value will trigger a cost increment when \( A_4 \) performs an assignment. Therefore \( h(v_4) = 1 \) is an admissible lower bound of the cost of the constraints between this value and lower priority agents. Since \( v_4 \) does not conflict with assignments on the CPA, \( f(v_4) = 1 \) as well. \( f(v_5) = 3 \) because this assignment conflicts with the assignments on the CPA and in addition conflicts with all the values of the two remaining agents.

Since \( h(v) \) takes into account only constraints of \( A_i \) with lower priority agents \( (A_j \text{ s.t. } j > i) \), unassigned lower priority agents do not need to estimate their cost of constraints with \( A_i \). Therefore, these estimations can be accumulated and summed up by the agent which initiated the forward bounding process to compute a lower bound on the cost of a complete assignment extended from the CPA. Asynchronous forward bounding enables agents an early detection of partial assignments that can not be extended into complete assignments with cost smaller than the known upper bound, and initiate backtracks as early as possible.

The AFB algorithm is run on each of the agents in the DisCOP. Each agent first calls the procedure \( init \) and then responds to messages until it receives a TERMINATE message. The algorithm is presented in Figure 2.

3.1 Algorithm description

The algorithm starts by each agent calling \( init \) and then awaiting messages until termination. At first, each agent updates \( B \) to be the cost of the best full assignment found so far and since no such assignment was found, it is set to infinity (line 1). Only the first agent \( (A_1) \) creates an empty CPA and then begins the search process by...
procedure init:
  1. $B \leftarrow \infty$
  2. if ($A_i = A_j$)  
     3. generate_CPA()
     4. assign_CPA()

when received (FB_CPA, A_j, PA)
  5. $f \leftarrow$ estimation based on the received PA.
  6. send (FB_ESTIMATE, $f$, PA, $A_i$) to $A_j$

when received (CPA_MSG, PA)
  7. CPA $\leftarrow$ PA
  8. TempCPA $\leftarrow$ PA
  9. if TempCPA contains an assignment to $A_i$, remove it
  10. if (TempCPA.cost $\geq B$)
      11. backtrack()
  12. else
     13. assign_CPA()

when received (FB_ESTIMATE, estimate, PA , A_j)
  14. save estimate
  15. if ( CPA.cost + all saved estimates ) $\geq B$
  16. assign_CPA()

when received (NEW_SOLUTION, PA)
  17. B_CPA $\leftarrow$ PA
  18. $B \leftarrow$ PA.cost

procedure assign_CPA:
  19. clear estimations
  20. if CPA contains an assignment $A_i = v$, remove it
  21. iterate (from last assigned value) over $D_i$ until found
     $v \in D_i$ s.t. CPA.cost + $f(v) < B$
  22. if no such value exists
     23. backtrack()
  24. else
     25. assign $A_i = v$
     26. if CPA is a full assignment
     27. broadcast (NEW_SOLUTION, CPA )
     28. $B \leftarrow$ CPA.cost
     29. assign_CPA()
  30. else
     31. send(CPA_MSG, CPA) to $A_{i+1}$
     32. forall $j > i$
     33. send(FB_CPA, $A_i$, CPA) to $A_j$

procedure backtrack:
  34. clear estimates
  35. if ($A_i = A_j$)
     36. broadcast(TERMINATE)
  37. else
     38. send(CPA_MSG, CPA) to $A_{i-1}$

Figure 2. The procedures of the AFB Algorithm

calling assign_CPA (lines 3-4), in order to find a value assignment
for its variable.

An agent receiving a CPA (when received CPA_MSG), first makes
sure it is relevant. The time stamp mechanism suggested by [13] is
used to determine the relevance of the CPA.

If the CPA’s time-stamp reveals that it is not the most up to date
CPA, the message is discarded. In such a case, the agent processing
the message has already received a message implying that an assign-
ment of some agent which has a higher priority than itself, has been
changed. When the message is not discarded, the agent saves the re-
ceived PA in its local CPA variable (line 7). Then, the agent checks
that the received PA (without an assignment to its own variable) does
not exceed the allowed cost $B$ (lines 8-10). If it does not exceed the
bound, it tries to assign a value to its variable (or replace its existing
assignment in case it has one already) by calling assign_CPA (line
13). If the bound is exceeded, a backtrack is initiated (line 11) and
the CPA is sent to a higher priority agent, since the cost is already
too high (even without an assignment to its variable).

Procedure assign_CPA attempts to find a value assignment, for
the current agent, within the bounds of the current CPA. First, es-

timates related to prior assignments are cleared (line 19). Next, the
agent attempts to assign every value in its domain if it did not already
try. If the CPA arrived without an assignment to its variable, it tries
every value in its domain. Otherwise, the search for such a value is
continued from the value following the last assigned value. The as-
signed value must be such that the sum of the cost of the CPA and the
lower bound of the cost increment caused by the assignment will not
exceed the upper bound $B$ (lines 20-22). If no such value is found,
then the assignment of some higher priority agent must be altered,
and so backtrack is called (line 23). Otherwise, the agent assigns the
selected value on the CPA.

When the agent is the last agent ($A_n$), a complete assignment has
been reached, with an accumulated cost lower than $B$, and it is broad-
cast to all agents (line 27). This broadcast will inform the agents of
the new bound for the cost of a full assignment, and cause them to
update their upper bound $B$.

The agent holding the CPA ($A_n$) continues the search, by updating
its bound $B$, and calling assign_CPA (line 29). The current value will
not be picked by this call, since the CPA’s cost with this assignment
is now equal to $B$, and the procedure demands the cost to be lower
than $B$. So the agent will continue the search, testing other values,
and backtracking in case they do not lead to further improvement.

When the agent holding the CPA is not the last agent (line 30), the
CPA is sent forward to the next unassigned agent, for additional
value assignment (line 31). Concurrently, forward bounding requests
(i.e. FB_CPA messages) are sent to all lower priority agents (lines
32-33).

An Agent receiving a forward bounding request (when received
FB_CPA) from agent $A_j$, again uses the time-stamp mechanism to
ignore irrelevant messages. Only if the message is relevant, then the
agent computes its estimate (lower bound) of the cost incurred by the
lowest cost assignment to its variable (line 5). The exact computation
of this estimation was described above (it is the minimal $f(v)$ over
all $v \in D_i$). This estimation is then attached to the message and sent
back to the sender, as a FB_ESTIMATE message.

An agent receiving a bound estimation (when received
FB_ESTIMATE) from a lower priority agent $A_j$ (in response to a
forward bounding message) ignores it if it is an estimate to an al-
ready abandoned partial assignment (identified using the time-stamp
mechanism). Otherwise, it saves this estimate (line 14) and checks if
this new estimate causes the current partial assignment to exceed the
bound $B$ (line 15). In such a case, the agent calls assign_CPA (line
16) in order to change its value assignment (or backtrack in case a
valid assignment cannot be found).

The call to backtrack is made whenever the current agent cannot
find a valid value (i.e. below the bound $B$). In such a case, the agent
clears its saved estimates, and sends the CPA backwards to agent
$A_{i-1}$ (line 38). If the agent is the first agent (nowhere to backtrack
to), the terminate broadcast ends the search process in all agents (line 36). The algorithm then reports that the optimal solution has a cost of $B$, and the full assignment with such cost is $B_CP_A$.

**4 Correctness of AFB**

In order to prove correctness for AFB two claims must be established. First, that the algorithm terminates and second that when the algorithm terminates its global upper bound $B$ is the cost of the optimal solution. To prove termination one can show that the AFB algorithm never goes into an endless loop. To prove the last statement it is enough to show that the same partial assignment cannot be generated more than once.

**Lemma 1** The AFB algorithm never generates two identical CPAs.

Assume by negation that $A_i$ is the highest priority agent (first in the order of assignments) that generates a partial assignment CPA for the second time. The replacement of an assignment can only be triggered by one of two messages arriving at $A_i$ from a lower priority agent $A_j$ ($j > i$). Either a backtrack CPA message, or a $FB_{ESTIMATE}$ message. In the first case the next assignment on the CPA will be generated by the procedure assign_CPA. Each of the values in the domain of $A_i$ is considered exactly once. When the agent’s domain is exhausted the agent backtracks and under the above assumption will never receive the same partial assignment again. If the received message is an estimate that clashes with the upper bound (e.g. the second case), a new CPA is generated. The generated CPA is a clone of the last CPA the agent received from a higher priority agent. Only values which were not considered on the previous CPA are left in its current domain. Therefore, the situation with the new CPA is similar to the first case. Termination follows immediately from Lemma 1.

Next we prove that on termination, the complete assignment, corresponding to the optimal solution is in $B_CP_A$ (see Figure 2). There is only one point of termination for the AFB algorithm, in procedure backtrack. So, one needs to prove that during search no partial assignment that can lead to a solution of lower cost than $B$ is discarded. But, this fact is immediate, because the only place in the code where values are discarded is in the third line of procedure assign_CPA (line 21). Within this procedure, values are discarded only when the calculated lower bound of the value being considered is higher than the current bound on the cost of a global solution. Clearly, this cannot lead to a discarding of a lower cost global solution.

One still needs to show that whenever the algorithm calls the procedure assign_CPA, it does not loose a potential lower cost solution. There are altogether 4 places in the algorithm, where a call to procedure assign_CPA is made. One is in procedure init, which is irrelevant. The three relevant calls are in the code performed when receiving a CPA or receiving a $FB_{ESTIMATE}$, and in the procedure assign_CPA itself.

The third case is trivially correct. Before calling the procedure the global bound $B$ is updated and the corresponding complete solution is stored. Consequently, the current solution is not lost. The first two calls (see Figure 2) appear in the last lines of the procedures processing the two messages. When processing a $FB_{ESTIMATE}$ message, the call to assign_CPA happens after the lower bound of the current value has been tested to exceed the global bound $B$ (line 15). This is correct, since the current partial solution cannot be extended to a lower cost solution. The last call to assign_CPA occurs in the last line of processing a received CPA message. Clearly, this call extends a shorter partial solution and does not discard a value of the current agent. This completes the correctness proof of the AFB algorithm.

**5 Experimental Evaluation**

All experiments were performed on a simulator in which agents are simulated by threads which communicate only through message passing. The Distributed Optimization problems used in all of the presented experiments are random Max-DisCSPs. Max-DisCSP is a subclass of DisCOP in which all constraint costs (weights) are equal to one [12]. The network of constraints, in each of the experiments, is generated randomly by selecting the probability $p_1$ of a constraint among any pair of variables and the probability $p_2$, for the occurrence of a violation (a non zero cost) among two assignments of values to a constrained pair of variables. Such uniform random constraints networks of $n$ variables, $k$ values in each domain, a constraints density of $p_1$ and tightness $p_2$ are commonly used in experimental evaluations of CSP algorithms (cf. [15]). Max-CSP was used in experimental evaluations of constraint optimization problems (COPs) by [3, 6]. Other experimental evaluations of DisCOPs include graph coloring problems [12, 18], which are a subclass of Max-DisCSP.

In order to evaluate the performance of distributed algorithms, two independent measures of performance are commonly used - run time, in the form of non-concurrent steps of computation, and communication load, in the form of the total number of messages sent [7, 17]. We use the method described in [10] for counting non-concurrent computational steps.

AFB is compared with the ADOPT algorithm, which is the state-of-the-art algorithm for DisCOPs. Our implementation of ADOPT is based on [12] as well as open code published by the designer of ADOPT, Jay Modi.

The ADOPT algorithm (as described in [12]) starts by first constructing a pseudo-tree of the agents. After this initial construction, each agent informs agents below it in the tree of its current value assignment. Each agent is responsible for reporting to its parent the optimal assignment cost of the sub problem for which it is a root.

The algorithm is asynchronous, therefore each agent reports only lower and upper bounds on its sub-problem. As the sub-problem is explored, these bounds are refined until eventually, the lower bound and upper bound become equal. At that point, the sub-problem is optimally solved [12].

In order to avoid thrashing through redundant assignment replacing, ADOPT uses a “threshold splitting” mechanism. Our implementation uses the same heuristic for threshold splitting used in the open code implementation (the “splitThresholdOneChild” heuristic). The construction of the pseudo tree is generated also according to the original implementation, using the well known DFS algorithm.

To help reduce the number of messages, we improved ADOPT’s implementation, having each agent read its entire mailbox, process all awaiting messages and only then send the messages needed. A similar improvement was done for the ABT algorithm [17, 19, 1]. In our experimental evaluation, we considered the processing of all these messages as a single computational step.

Synchronous Branch and Bound (SBB) is a simple algorithm, that simulates centralized Branch & Bound in a distributed environment [17]. A single CPA which carries the search state, and is passed between agents is all that is needed.

Figure 3 presents the average run-time in number of computation steps, of the three algorithms AFB, ADOPT and SBB on Max-DisCSPs with $n = 10$ agents, domain size $k = 10$, and a constraint density of $p_1 = 0.4$. The value of $p_2$ was varied between 0.4 and
0.98, to cover all ranges of problem difficulty [15]. It is clear in Figure 3 that as the tightness becomes larger and the problem harder, AFB outperforms ADOPT which outperforms SBB. For tightness values that are higher than 0.9 AFB demonstrates a phase transition, while ADOPT’s and SBB’s run-time keeps growing. The behavior of AFB is very similar to that of lookahead algorithms on centralized Max-CSPs [3, 4, 6]. The run-time of ADOPT ad SBB increases exponentially fast, so that on the very extreme values of \( p_2 \) it simply did not terminate in a reasonable time and their execution was terminated. Figure 4 shows the total number of messages sent by the three algorithms. AFB clearly outperforms ADOPT which outperforms SBB in this measure.

6 Conclusion

The present paper proposes a DisCOP algorithm that performs Forward-Bounding asynchronously. Extended consistency maintenance procedures are performed concurrently. The results presented in Section 5 demonstrate the importance of consistency maintenance for distributed search. The concurrent form of extended lookahead (forward bounding) prevents an exponential growth in run-time for the tighter problem instances. This is similar to the centralized case of Max-CSPs [3, 6]. The run-time performance of AFB is far better than ADOPT on tight distributed Max-DisCSPs. The advantage of AFB over ADOPT in overall network load is even more pronounced (Figure 4).

Recent studies of centralized constraints optimization problems have shown that the use of extended consistency maintenance procedures improves the performance of simple Branch and Bound [4, 6]. In particular, the hardest problem instances (with high \( p_2 \) value) show a phase transition, in which run-time decreases [3, 4, 5, 6].

Previous studies of distributed COPs have shown many advantages of the ADOPT algorithm over SBB, which is the naive algorithm for solving these problems [12]. Previous experimental evaluations of ADOPT measured its scalability, by increasing the number of variables/agents (12, 9, 8). The behavior of ADOPT on the hardest (tightest) Max-DisCSP instances was not explored. The experiments in the present study measure the algorithm’s performance over varying problem difficulty levels. The exponential growth of concurrent run-time of ADOPT for very hard and tight problem instances is therefore, only apparent in the present study.

REFERENCES