Asynchronous Forward-Bounding for Distributed Constraints Optimization

Amir Gershman, Amnon Meisels and Roie Zivan
{amirger,am,zivanr}@cs.bgu.ac.il

Department of Computer Science,
Ben-Gurion University of the Negev,
Beer-Sheva, 84-105, Israel

Abstract. A new search algorithm for solving distributed constraint optimization problems (*DisCOP*s) 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. Experimantal evaluation shows *AFB* outperforms synchronous branch and bound by many orders of magnitude, and reveals a phase transition as the tighness of the problem increases. This demonstrates that asynchronous forward-bounding has an analogous effect to the phase transition that has been observed when local consistency maintainance is applied to MaxCSPs.

Key Words: Distributed Optimization, Constraints, Distributed Search

1 Introduction

The Distributed Constraint Optimization Problem ($DisCOP$) is a general framework for distributed problem solving that has a wide range of applications in MultiAgent Systems and has generated significant interest from researchers [MSTY05,ZXWW05]. Distributed constraints optimization problems (*DisCOP*s) 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. [MSTY05,PF04]). To achieve this goal, agents exchange messages among them, to check the cost of their proposed assignments against constraints among variables that belong to different agents.

Distributed COPs are an elegant model for many every day 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 [SGM96,KM05].

Several distributed search algorithms for *DisCOP*s have been proposed in recent years [Yok00,MSTY05,PF04]. All of these algorithms process assignments of agents asynchronously. The present paper proposes a new distributed search algorithm for *DisCOP*s, *Asynchronous Forward-Bounding (AFB)*. In the *AFB* algorithm agents assign their variables and generate a partial solution sequentially. The innovation of the proposed algorithm lies in propagating partial solutions asynchronously. Propogation of partial solutions enables asynchronous updating of bounds on their cost, and early detection of a need to backtrack, hence the algorithm's name *AFB*.

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, 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 sum on the costs of broken constraints it includes. Besides the current assignment cost, the agents maintain on a CPA a *lower bound* which is updated according to information they receive from yet unassigned agents. Each agent which receives the CPA, adds assignments of its local variables to the partial assignment on the received CPA, if an assignment with a lower bound smaller than the current global upper bound can be found. Otherwise, it backtracks by sending the *CPA* to a former agent to revise its assignment.

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. The assigning agent will recieve these estimations asynchronously over time and use them to update the lower bound of the *CPA*.

Gathering updated lower bounds from future assigning agents, may enable an agent to discover that the lower bound of the *CPA* it sent forward is higher than the current upper bound (i.e. inconsistent). This discovery triggers the creation of a new CPA which is a copy of the CPA it sent forward. The agent resumes the search by trying to replace its inconsistent assignment. The time stamp mechanism suggested by [NSHF04] is used by agents to determine the most updated CPA and to discard obsolete $CPAs$.

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 (*SBnB*) on randomly generated *DisCOP*s. *AFB* outperforms *SBnB* 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 constraints checks 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 reported for extended local consistency maintainance on MaxCSPs [LM96,LMS99].

2 Distributed Constraint Optimization

A distributed constraint optimization problem - ***DisCOP*** is composed of a set of k agents $A_1, A_2, ..., A_k$. Each agent A_i contains a set of constrained variables $X_{i_1}, X_{i_2}, ..., X_{i_{n_i}}$. Constraints or **relations** R are subsets of the *Cartesian Product* of the domains of the constrained variables. For a set of constrained variables $X_{i_k}, X_{j_l}, ..., X_{m_n}$, with domains of values for each variable $D_{i_k}, D_{j_l}, ..., D_{m_n}$, the constraint is defined as $R \subseteq D_{i_k} \times D_{j_l} \times ... \times D_{m_n}$. A **binary constraint** R_{ij} between any two variables X_j and X_i is a subset of the *Cartesian Product* of their domains; $R_{ij} \subseteq D_j \times D_i$. Each constraint R_{ij} in a constraint optimization problem, is associated with a weight w_{ij}.

In a distributed constraint optimization problem *DisCOP*, the agents are connected by constraints between variables that belong to different agents [MSTY05,SGM96]. An assignment (or a label) is a pair $< var, val >$, where var is a variable of some agent and val is a value from var's domain that is assigned to it. A *partial assignment* is a set of assignments of values to a set of variables. The cost of an assignment to all variables $< X_{i1}, Val_{i1} >, < X_{i2}, Val_{i2} >, \ldots, < X_{in}, Val_{in} >$ is the sum of the weights of all violated constraints. A **solution** P to a *DisCOP* is an assignment to all variables of all agents that has the lowest cost.

In this paper, we will assume 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 [Yok00,MSTY05]. Furthermore, we assume a static final order on the agents, known to all agents participating in the search process [Yok00]. These assumptions are commonly used by DisCSP and DisCOP algorithms [Yok00,MSTY05].

3 Asynchronous Forward Bounding - AFB

In the *AFB* algorithm only one agent perfroms an assignment on a CPA at a time. Partial assignments are propageted forward asynchronously. Each such assignment is processed by multiple agents concurrently. Each unassigned agent computes a lower bound of 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 cheapest full assignment found so far.

Agents assign their variables only when they hold the current partial assignment (*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.

Forward bounding is performed as follows. 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 for its variable to the CPA. This estimated cost is sent back to the agent which sent the *FB_CPA* message via *FB_ESTIMATE* messages.

More formally the estimated cost is computed as follows. denote by $\mathbf{cost}((i,v),(j,u))$ the cost of assigning $A_i = v$ and $A_j = u$. If $v \in D_i$ then we deonte by $h_j(v) = min(cost(i,v),(j,u))$ s.t. $u \in D_j$. We define $h(v)$, 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 $j > i$. Notice this bound can be computed once per agent, since it is independent of the assignments of higher prority agents.

When recieving a *FB_CPA* message, the agent computes the cost increment to the CPA from assigning the value v to its variable. To this cost the agent adds $h(v)$ since this is the minimal cost which will be added from conflicts between this assignment and assignments of lower priorty agents (who are unassigned on the *FB_CPA*). This sum is denoted by $f(v)$. The estimate returned by this agent (on a *FB_ESTIMATE* message) is the minimal $f(v)$, s.t. $v \in D_i$, since this is a lower bound of the cost which will be added to the *CPA* cost once this agent extends the *CPA* with an assignment to its variable.

The Agent which initiated the forward bounding process, receives such estimations (lower bounds) from lower prority agents and can use them to compute a lower bound on the cost of a complete assignment extended from the *CPA*. Therefore, 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.

All *CPA*s are time-stamped by each agent. The time stamp is a concatenation of the running assignment counters of all agents, from high to low priority. This unique time-stamp enables a receiving agent to compare the message received to its own time-stamp, discovering irrelevant partial assignments (i.e. *CPA*s that contain an assignment that has been replaced already). Agents that receive obsolete *CPA*s (or *FB_CPA*s) simply discard them.

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 an order to terminate. The algorithm uses the following objects and messages:

- *CPA (current partial assignment)*: A message that carries the currentl partial assignment. A *CPA* is composed of triplets of the form $< A, X, V >$ where A is the agent that owns variable X and V is the value that was assigned to X by A. Each *CPA* contains a time-stamp that is updated by each agent that assigns its variables on the *CPA*, and an accumulated *cost* of all conflicts between all assignments written on the *CPA*.
- *FB_CPA* : A message that is an exact copy of a *CPA*. Every agent that assigns its variables on a *CPA*, creates an exact copy in the form of a *FB_CPA* (with the same time-stamp) and sends it forward to all unassigned agents.
- *FB_ESTIMATE*: A message carrying the estimate calculated by some unassigned agent in response to a *FB_CPA*. The estimate is a lower bound on the cost which will be added to the *CPA* by assigning a value to the unassigned agent.
- *NEW_SOLUTION*: A message containing a full assignment (and it's accumulated cost of constraints). This full assignment becomes the 'best known full assignment found so far' and it's cost is used as the upper bound in the branch and bound process.

– *STOP*: A messages used to terminate the run of all agents.

3.1 Algorithm description

The procedures of the algorithm ***AFB*** that deal with the four types of messages are presented in figure 1. The procedure for assigning values is presented in figure 3 and the rest of the procedures and functions are presented in figure 2.

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 cheapest full assignment found so far and since no such assignment was found, it is set to infinity (line 1). Next, the agents initialize their local timestamps and cost estimations to zero (lines 3-5). Afterwards, each agent computes $h(v)$ for every value v in its domain (line 6). Only the first agent (A_1) creates an empty CPA and then begins the search process by calling $assign_CPA$ (lines 7-9), in order to find a value assignment for its variable.

An agent receiving a CPA (**when received** CPA_MSG), first makes sure that the CPA was not already abandoned by higher priorty agents. It does so by checking that the timestamp on the message is greater or equal lexicographically to its own timestamp. If this is not so, the message is discarded (line 4). In case the received timestamp is greater than its own timestamp, the agent updates its timestamp to the greater value (line 3). Then, the agent checks that the received CPA (without an assignment to its variable) is within the cost bound. If this is the case, it calls $assign_CPA$ (line 10). Otherwise, since any assignment could only increase the cost, the received partial assignment must be changed. Backtrack is called (line 8) and the CPA is sent to some higher priority agent to change his assignment.

The procedure **assign_CPA** attempts to find a value assignment, for the current agent, within the bounds of the current CPA. First, estimates of prior assignments are cleared (line 1). Next, the agent attempts to assign every value in its domain it did not already try. Either a value within the bounds of the CPA is found (line 9), or all values are determined to be nogoods. When the agent cannot find a value assignment, backtrack is called (line 12). Otherwise, the agent updates both its local timestamp and the one on the CPA (lines 14-16). When the agent is the last agent, a complete assignment has been reached, with an accumulated cost lower than B, and the cost of the current assignment becomes the new upper bound B by calling $upon_solution_found$ (line 18). When the agent is not the last agent, the CPA is sent forward to the next unassigned agent, for additional value assignment (line 20). Concurrently, forward bounding requests (i.e. FB_CPA messages) are sent to all lower priority agents (lines 21-22).

An Agent receiving a forward bounding request (**when received** FB_CPA) from agent A_j, discards or updates it according to the relevance of its timestamp (lines 3, 4). Next, the agent computes its estimate of the cost incurred by the best assignment of its variable and sends it back to A_j. The estimation is calculated in function $estimate_added_cost$ in the following way. First $f(v)$ is calculated (as described before) for all values $v \in D_i$ (lines 1-2). Then the minimal $f(v)$ is found and returned. This value is a lower bound on the cost which will be added by A_i when it adds an assignment to its variable to the current CPA.

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

```
procedure init:
1.   B ← ∞
2.   time ← 0
3.   foreach j=1 to n
4.      estimates[j] ← 0
5.      timestamp_threshold[j] ← 0
6.   compute_h()
7.   if (A_i = A_1)
8.      generate_CPA()
9.      assign_CPA()

when received (STOP)
1.   Terminate execution

when received (FB_CPA, A_j, received_CPA)
1.   temp_CPA ← received_CPA
2.   switch (compare ( temp_CPA.timestamp, timestamp_threshold, i − 1 ))
3.      "bigger": timestamp_threshold ← temp_CPA.timestamp
4.      "smaller": return
5.   f ← estimate_added_cost()
6.   send (FB_ESTIMATE, f, temp_CPA.timestamp, A_i) to A_j

when received (CPA_MSG, received_CPA)
1.   temp_CPA ← received_CPA
2.   switch (compare (temp_CPA.timestamp, timestamp_threshold, i ))
3.      "bigger": timestamp_threshold ← temp_CPA.timestamp
4.      "smaller": return
5.   CPA ← temp_CPA
6.   remove_my_value_and_update_cost(temp_CPA)
7.   if (temp_CPA.cost ≥ B)
8.      backtrack()
9.   else
10.     assign_CPA()

when received (NEW_SOLUTION, received_CPA)
1.   B_CPA ← received_CPA
2.   B ← B_CPA.cost

when received (FB_ESTIMATE, estimate, timestamp, A_j)
1.   switch (compare(timestamp, timestamp_threshold, i))
2.      "smaller": return
3.   estimates[j] ← estimate
4.   if ( CPA.cost + Σ_{i+1≤j≤n} estimates[j] ≥ B )
5.      assign_CPA()
```

Fig. 1. The receiving functions of the AFB Algorithm

```
procedure backtrack:
1.  remove_my_value_and_update_cost(CPA)
2.  timestamp_threshold[i] + +
3.  CPA.timestamp[i] ← 0
4.  clear(estimates)
5.  if (A_i = A_1)
6.     broadcast(STOP)
7.     Terminate execution
8.  else
9.     send(CPA_MSG, CPA) to A_{i−1}

function estimate_added_cost:
1.  forall v ∈ D_i
2.     f[v] ← cost_of_constraints(v, temp_CPA) + h[v]
3.  return min(f[v])

procedure compute_h:
1.  forall v ∈ D_i
2.     h[v] ← 0
3.     forall j > i and j ≤ n
4.        h[v] ← h[v] + min(cost_of_constraints(v, u))
                           s.t. u ∈ D_j

procedure upon_solution_found:
1.  broadcast(NEW_SOLUTION, CPA)
2.  B_CPA ← CPA
3.  B ← CPA.cost
4.  assign_CPA()

function compare (timestamp1, timestamp2, upto):
1.  for j ← 1 to upto do
2.     if timestamp1[j] < timestamp2[j] return "smaller"
3.     if timestamp1[j] > timestamp2[j] return "bigger"
4.  return "equal"
```

Fig. 2. AFB functions

```
procedure assign_CPA:
1.  clear(estimates)
2.  if (my_current_value(CPA, i) = −1)
3.      time ← 0
4.  found ← false
5.  exshasted_domain ← (my_current_value(CPA, i) = last value in D_i)
6.  while (¬ found and ¬ exshusted_domain)
7.      next_value_and_update_cost(CPA)
8.      v ← my_current_value(CPA, i)
9.      found ← (CPA.cost + h[v] < B)
10.     exshusted_domain ← (my_current_value(CPA, i) = last value in D_i)
11. if (¬found)
12.     backtrack()
13. else
14.     time++
15.     CPA.timestamp[i] ← time
16.     timestamp_threshold ← CPA.timestamp
17.     if (i = n)
18.         upon_solution_found()
19.     else
20.         send(CPA_MSG, CPA) to A_{i+1}
21.     forall j > i
22.         send(FB_CPA, A_i, CPA) to A_j
```

Fig. 3. AFB - the assign procedure

estimate to an already abandoned partial assignment (lines 1-2). Otherwise, it saves this estimate (line 3) and checks if this new estimate causes the current partial assignment to exceed the bound B (line 4). In such a case, the agent calls $assign_CPA$ (line 5) in order to change its value assignment (or backtrack in case a valid assignment cannot be found).

Once the last agent is holding a complete assignment with a better cost than B, it calls **upon_solution_found** and broadcasts this assignment to all other agents (line 1). It then updates its value of B (line 3) and calls $assign_CPA$ (line 4). $assign_CPA$ will cause this agent to search for another value, that will produce an accumulated cost smaller than the new value of B. If the agent can find such a value, it will discover a new (lower cost) full assignment and call $upon_solution_found$ once more. Otherwise, it will backtrack the CPA to higher priority agents in order for them to change their assignment.

An agent receiving a new full assignment with a smaller cost than B (**when received** $NEW_SOLUTION$) simply updates its value B to the cost of the received assignment.

A 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 removes its value from the CPA (line 1), updates its timestamp (line 2) and the timestamp on the CPA (line 3). Then it clears its estimates for the new CPA (line 4) and sends the CPA backwards to agent A_{i-1} (line 9). In case there is no previous agent, the algorithm has explored the entire search space, and sends out a terminate command to all agents. The algorithm then reports that the optimal solution is of cost B, and a full assignment of such cost is B_CPA.

4 Correctness of AFB

In order to prove correctness for AFB one must show that when the algorithm terminates its global upper bound B is the cost of the optimal solution. The complete assignment, corresponding to the optimal solution are in B_CPA (see Figure 1). 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 lines 7-9 of procedure *assign_CPA*. 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 (line 9 of Figure 3). 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 $upon_solution_found$.

The third case is trivially correct. The procedure $upon_solution_found$ updates the global bound B and stores the corresponding complete solution. Consequently, the current solution is not lost. The first two calls (see Figure 1) 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. 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

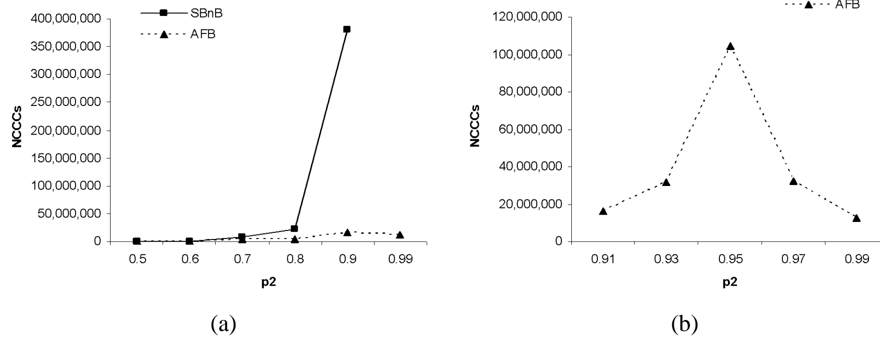
All experiments use a simulator in which agents are simulated by threads which communicate only through message passing. All experiments are performed on random MaxDisCSPs. MaxDisCSP is a subclass of weighted DisCSP in which all constraint weights are eqaul to one. The network of constraints, in each 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 occurrance of a violation 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. [Pro96,Smi96]). MaxCSP was used in experimental evaluations of constraint opimization problems by [LS04]. Other experimental evaluations of DisOPTs include max graph coloring problems [MSTY05,ZXWW05], which are a subclass of MaxDisCSP.

The experimental setup includes problems generated with 10 variables ($n = 10$) and 10 values ($k = 10$). The experiments include $MaxDisCSPs$ with two different network density values $p_1 = 0.4$ and $p_1 = 0.7$. The value of p_2 was varied between 0.5 and 0.99, to cover all ranges of problem difficulty [Pro96].

In order to evaluate the performance of distributed algorithms, two independent measures of performance are commonly used - run time in the form of steps of computation [Lyn97,Yok00] and communication load in the form of the total number of messages sent [Lyn97]. To take into account the local computation performed by agents in each step, computational cost can be evaluated in terms of non concurrent constraints checks. The evaluation of the computational effort of distributed algorithms has to take concurrency into account. Non-concurrent constraint checks, in systems with no message delay, are counted by a method similar to that of Lamport [Lam78,MRKZ02,ZM04]. Every agent holds a counter of constraint checks. Every message carries the value of the sending agent's counter. When an agent receives a message it updates its counter to the largest value between its own counter and the counter value carried by the message. By reporting the cost of the search as the largest counter held by some agent at the end of the search, a concurrent search effort that is close to Lamport's logical time [Lam78] is achieved.

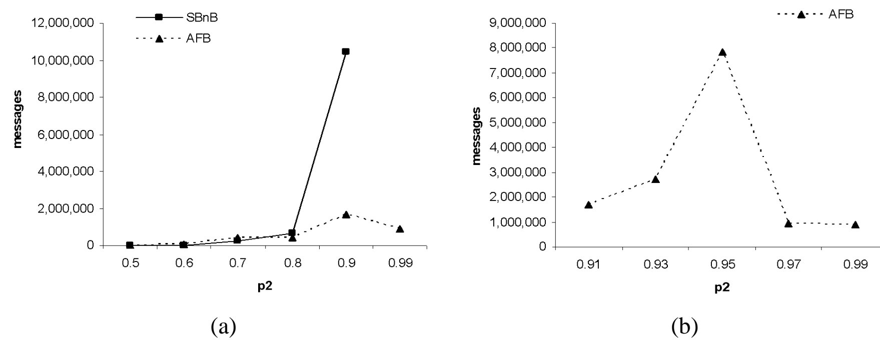
The total number of messages sent during the run of the algorithm is a common measure of network load for distributed algorithms [Lyn97].

To evaluate the performance of *AFB*, it is compared to the synchronous branch and bound algorithm (SBnB). SBnB is simple branch and bound algorithm, in which only one agent performs an assignment at a time. A single *CPA* message is exchanged between the agents. The *CPA* is replaced, when its cost exceeds that of the best known



(a) (b)

Fig. 4. (a) Non-concurrent constraints checks (NCCCs) performed by AFB and SBnB for low density ($p_1 = 0.4$) MaxDisCSP. (b) A closer look at $p_2 > 0.9$



(a) (b)

Fig. 5. (a) Number of messages sent by AFB and SBnB for low density MaxDisCSP ($p_1 = 0.4$). (b) A closer look at $p_2 > 0.9$

full assigmment found so far. In such a case, the last assignment is unmade, and another value is selected instead.

Figure 4 presents the results in run-time of AFB and $SBnB$ for low density $MaxDisCSP$s. The RHS of the figure presents a detailed view ("zoom in") of the run-time behaviour for the tighter problem instances. AFB outperforms $SBnB$ in terms of run-time. The tighter the problems become, the greater the performance improvement of AFB. For p_2=0.9, AFB's speedup is by a factor greater than 20. A phase-transition is aparent at very high values of p_2. This phase-transition was reported for centralized MaxCSPs by [LM96,LMS99] and was associated with extended maintainance of local consistency. The phase-transition can be explained by the fact that AFB performs a form of local consistency branch and bound (in its asynchronous propogation of lower bounds).

Figure 5 presents the network load of AFB and $SBnB$ for low density $MaxDisCSP$s. AFB outperforms $SBnB$ in terms of network load as well. Once again, the tighter

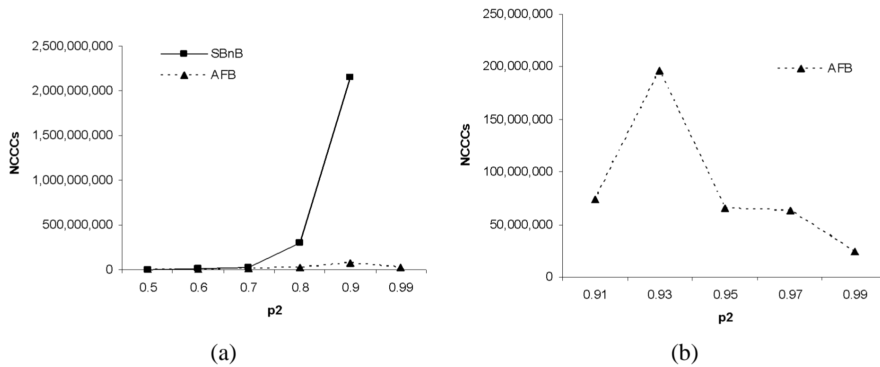


Fig. 6. (a) Non-concurrent constraints checks (NCCCs) for high density ($p_1 = 0.7$) MaxDisCSPs. (b) A closer look at $p_2 > 0.9$

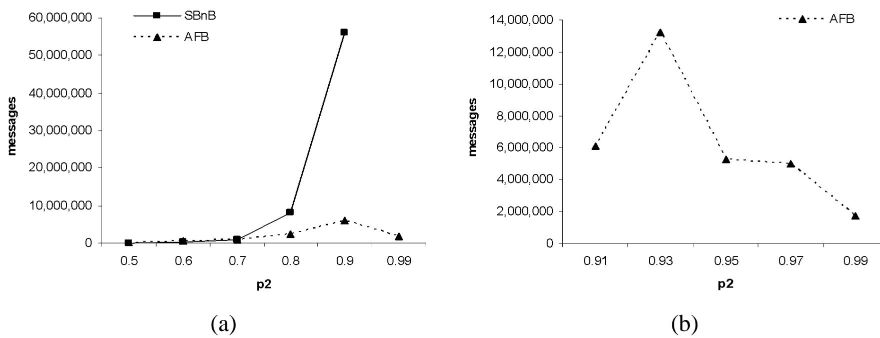


Fig. 7. (a) Number of messages sent by AFB and SBnB for high density MaxDisCSP ($p_1 = 0.7$). (b) A closer look at $p_2 > 0.9$

the problems, the greater the difference between the two algorithms, and for p_2=0.9 a speedup by a factor of 6 is achieved.

Figures 6 and 7 present similar results for higher density $MaxDisCSP$s. AFC outperforms $SBnB$ and the difference increases as problems become tighter.

6 Discussion

Previous work on centralized constraint optimization problems has shown that algorithms which implement the branch and bound strategy without maintaining some form of local consistency during the search, only increase in run time as problems become tighter [WF93,LM96]. Algorithms which maintain some form of extended local consistency during search (such as directional arc-consistency, AC*) report a phase-transition in performance, as the tightness of the problem increases beyond a certain threshold [LM96,LMS99,LS04]. Not all forms of local consistency produce a maximum of

run-time at higher density problems [LS04]. The conslusions of former works on centralized MaxCSPs suggest that constraints between unassigned agents must be taken into consideration in the maintance of local consistency in order to achieve this improved performance.

The algorithm presented in the present paper, attemps to extend previous work on centralized problems to the distributed problem world. The comparison to a branch and bound algorithm demonstrates the effectivness of this strategy in the distributed case. AFB offers a way to compute lower bounds of partial assignmetns concurrently and asynchronously (through the use of its forward bounding procedures) and thus maintains a form of local consistency. In AFB each agent computing a lower bound includes in its estimate both the cost of conflicts between assigned varaibles to its unassigned varaible, and estimates of costs among the unassigned varaibles. As shown in the experimental evaluation, AFB shows a phase-transition that is similar to former studies of MaxCSPs [LM96,LMS99,LS04].

The present work is not a simple extention of the centralized case because the local consistency maintainance is performed concurrently by all agents. It is clear that in the centralized case the computational effort to enforce local consistency after every assignemnt must produce some run-time overhead. This generates a balance between stronger methods for maintaining consistency and thus less nodes visited, and between the computational effort to enfornce this consistency at every node. This is not necesseraly the case for $DisCOP$s. AFB maintains local consistency concurrently by all agents, and advances the search process to perform the next assignment.

Other related work includes $ADOPT$ [MSTY05] which uses an "opportunistic best first value selection" and may backtrack before proving an assignment cannot lead to an optimal solution. $ADOPT$ remains complete through the use of a complex "backtrack threshold" mechanism which allows it to revisit previously abandoned assignments. The present paper presents a simple and efficient algorithm for solving $DisCOP$s without the need of such a complex mechanism, and without revisiting nodes. Due to the complex nature of $ADOPT$, its experimental comparison with AFB is currently being performed and will be reported in the near future.

It should be noted, that the heuristic used by AFB (i.e. the function $h(v)$) does not have to be the same as the one that was used in our implementaiton. In fact, any admissibile heuristic estimation on the lower bound of the cost of constraints between the assignment $A_i = v$ and lower priority agents can be used. The heuristic we used is similar to the way that NC* [LS04] performs local consistency.

References

[KM05] E. Kaplansky and A. Meisels. Distributed personnel scheduling: Negotiation among scheduling agents. *Annals of Operations Research*, 2005.

[Lam78] L. Lamport. Time, clocks, and the ordering of events in distributed system. *Communication of the ACM*, 2:95–114, April 1978.

[LM96] J. Larrosa and P. Meseguer. Phase transition in max-csp. In *Proc. ECAI-96*, Budapest, 1996.

[LMS99] J. Larrosa, P. Meseguer, and T. Schiex. Maintaining reversible dac for max-csp. *Artificial Intelligence*, 107:149–163, 1999.

[LS04] J. Larrosa and T. Schiex. Solving weighted csp by maintaining arc consistency. *Artificial Intelligence*, 159:1–26, 2004.

[Lyn97] N. A. Lynch. *Distributed Algorithms*. Morgan Kaufmann Series, 1997.

[MRKZ02] A. Meisels, I. Razgon, E. Kaplansky, and R. Zivan. Comparing performance of distributed constraints processing algorithms. In *Proc. AAMAS-2002 Workshop on Distributed Constraint Reasoning DCR*, pages 86–93, Bologna, July 2002.

[MSTY05] P. J. Modi, W. Shen, M. Tambe, and M. Yokoo. Adopt: asynchronous distributed constraints optimization with quality guarantees. *Artificial Intelligence*, 161:1-2:149–180, January 2005.

[NSHF04] T. Nguyen, D. Sam-Hroud, and B. Faltings. Dynamic distributed backjumping. In *Proc. 5th workshop on distributed constraints reasoning DCR-04*, Toronto, September 2004.

[PF04] A. Petcu and B. Faltings. A value ordering heuristic for distributed resource allocation. In *Proc. CSCLP04, Lausanne, Switzerland*, http://liawww.epfl.ch/Publications/Archive/Petcu2004.pdf, 2004.

[Pro96] P. Prosser. An empirical study of phase transitions in binary constraint satisfaction problems. *Artificial Intelligence*, 81:81–109, 1996.

[SGM96] G. Solotorevsky, E. Gudes, and A. Meisels. Modeling and solving distributed constraint satisfaction problems (dcsps). In *Constraint Processing-96*, pages 561–2, New Hamphshire, October 1996.

[Smi96] B. M. Smith. Locating the phase transition in binary constraint satisfaction problems. *Artificial Intelligence*, 81:155 – 181, 1996.

[WF93] R. J. Wallace and E. C. Freuder. Conjunctive width heuristics for maximal constraint satisfaction. In *Proc. AAAI-93*, pages 762–768, 1993.

[Yok00] M. Yokoo. Algorithms for distributed constraint satisfaction problems: A review. *Autonomous Agents & Multi-Agent Sys.*, 3:198–212, 2000.

[ZM04] R. Zivan and A. Meisels. Message delay and discsp search algorithms. In *Proc. 5th workshop on distributed constraints reasoning, DCR-04*, Toronto, 2004.

[ZXWW05] W. Zhang, Z. Xing, G. Wang, and L. Wittenburg. Distributed stochastic search and distributed breakout: properties, comparishon and applications to constraints optimization problems in sensor networks. *Artificial Intelligence*, 161:1-2:55–88, January 2005.