Asynchronous Branch & Bound and A* for DisWCSPs, with heuristic function based on Consistency-Maintenance

Marius-Călin Silaghi[†] and Jerry Landwehr[†] and Javier Bondia Larrosa[‡]
[†]*Florida Institute of Technology*
[‡]*Universitat Politècnica de Catalunya*

Abstract.
Distributed weighted constraint satisfaction problems occur, for example, when privacy requirements exclude the centralization of the problem description. New definitions of arc consistency (AC) and node consistency (NC*) for centralized weighted constraint satisfaction problems (WCSP) have been recently proposed and prove increased simplicity and effectiveness [4]. We show how they can be exploited in asynchronous search: (a) Weighted Consistency Labels (WCL) are introduced to represent costs inherent to each value of a variable in an explored subproblem. (b) A maximal propagation of the weights with NC* can be guaranteed by either (b') designating a single agent for performing the summation of the costs synthesized by different agents for the same value and subproblem, or (b") having all agents learn and separately sum up all such costs. (c) The obtained weights combined with asynchronous Branch & Bound (B & B) leads to a distributed equivalent of powerful centralized versions of B & B. (d) The heuristic function needed by asynchronous A* can also be extracted from such costs computed concurrently with the search.

1 Introduction

An agent's private requirements can often be formulated in a general framework such as constraint satisfaction problems (i.e. where everything is modeled by either variables, values, or constraints) and then can be solved with any of the applicable CSP techniques. But often one has to also find agreements with the other agents for a solution from the set of possible valuations of shared resources that satisfy her subproblem. The general framework modeling this kind of distributed combinatorial problems is called Distributed Constraint Satisfaction.

In practice one often meets optimization problems. The techniques developed for satisfaction problems have proven to be very useful when adapted to fit optimization problems (e.g. arc consistency maintenance led to PFC-MRDAC [5]). Two simpler and efficient techniques, W-AC and W-AC*, were recently developped for enforcing arc consistency in WCSPs [4].

Distributed Weighted CSPs (DisWCSPs) is a general formalism that can model negotiation problems and can quantify their privacy requirements [15, 2, 14]. Here it is shown how a basic technique for constraint satisfaction, namely maintainance of consistency can be combined with W-AC* and applied to asynchronous Branch & Bound and A* for optimization in DisWCSPs.

x_1/x_2	0	1	2	3
0	2	5	4	7
1	2	6	5	4
2	1	5	5	7
3	1	2	2	∞

Figure 1: Example of a constraint in a WCSP.

We first introduce the distributed Weighted Constraint Satisfaction Problem and the notion of arc consistency for weighted CSPs, W-AC*. The initial definition of W-AC* considered only binary constraints, but its extension to n-ary constraints is so straightforward that we present it directly. Next, we introduce the basic asynchronous backtracking algorithm for solving distributed CSPs, ABT. Three incremental modifications to ABT are used to introduce the proposed technique. A first small modification to ABT is ABT-cL, and is intended to allow us more flexibility for future modifications. The second modification shows how Branch & Bound can help ABT-cL to tackle distributed DisWCSPs. In the end, a set of alternative ways of maintaining arc consistency are analyzed and compared theoretically.

2 Distributed Weighted CSPs

CSP A *constraint satisfaction problem* (CSP) is defined by three sets: $(X,\ D,\ C)$. $X = \{x_1, ..., x_n\}$ is a set of variables and $D = \{D_1, ..., D_n\}$ is a set of domains such that x_i can take values only from D_i. $C = \{\phi_1, ..., \phi_c\}$ is a set of constraints such that ϕ_i is a predicate over an ordered subset X_i of the variables in X, $X_i \subseteq X$. An assignment is a pair $\langle x_i, v\rangle$ meaning that the variable x_i is assigned the value v. ϕ_i specifies the legality of each combination of assignments to the variables in X_i with values in their domains.

A tuple is an ordered set. The projection of the set of assignments in a tuple ϵ over a tuple of variables X_i is denoted $\epsilon|_{X_i}$. A solution of a CSP $(X,\ D,\ C)$ is a tuple of assignments $\epsilon*$ with one assignment for each variable in X (i.e. $\epsilon{*}{\in}D_1{\times}...{\times}D_n$) such that all the constraints $\phi_i \in C$ are satisfied by $\epsilon * |_{X_i}$.

Constraint Satisfaction Problems (CSPs) do not model optimization requirements. An extension allowing for modeling optimization concerns is given by Weighted CSPs.

Definition 1 (WCSP). *A Weighted CSP is defined by a triplet of sets (X, D, C) and a bound B. X and D are defined as in CSPs. In contrast to CSPs, C=$\{\phi_1, ..., \phi_c\}$ is a set of functions, $\phi_i : D_{i_1}{\times}...{\times}D_{i_{m_i}} \rightarrow \mathbb{N}^\infty$ where m_i is the arity of ϕ_i.*

Its solution is $\epsilon* = \underset{\epsilon\in D_1\times...\times D_n}{\operatorname{argmin}} \sum_{i=1}^{c} \phi_i(\epsilon|_{X_i})$, if $\sum_{i=1}^{c} \phi_i(\epsilon * |_{X_i}) < B$.

Example 1. *An example of a binary constraint in WCSPs is given in Figure 1. It is known that any maximization problem can be straightforwardly translated into a minimization problem. Weighted CSP can be also distributed.*

A Distributed CSP (DisCSP) is defined by four sets (A, X, D, C). A=$\{A_1, ..., A_n\}$ is a set of agents. X, D, C and the solution are defined like in CSPs except that $|C|$=$|X|$=$|A|$=n. Assignments for each variable x_i can be proposed only by A_i. Each constraint ϕ_i is defined only on x_i and its predecessors, and is known only by A_i.

Definition 2 (DisWCSP). *A Distributed Weighted CSP is defined by four sets (A, X, D, C) and a bound B. A, X, D are defined as for DisCSPs. In contrast to DisCSPs, C is a set of functions $\phi_i : D_{i_1} \times ... \times D_{i_{k_i}} \rightarrow \mathbb{N}^\infty$ defined on x_i and on some of its predecessors, and known only by A_i.*

Its solution is $\epsilon* = \underset{\epsilon \in D_1 \times ... \times D_n}{\text{argmin}} \sum_{i=1}^{c} \phi_i(\epsilon|_{X_i})$, if $\sum_{i=1}^{c} \phi_i(\epsilon * |_{X_i}) < B$.

Arc consistency We recall (see any basic AI manual like Russell&Norvig's [10]) that local arc/bound consistency is a property of a labeling of variables for a CSP. A *label* for x_i is nothing else than a set of values containing the possible valuations of x_i. In initial problem descriptions variables may also have in their domains values that no not appear in any solution. While such values add exponential complexity to systematic search techniques, some of them can be detected and eliminated with local observations on small subproblems bounded by a predefined size. These eliminations require an effort that is only polynomial in the size of the global problem. Therefore, recalculation/shrinking of labels based on local reasoning is a principled technique, very recommended, specially in its forms where cost complexity is of a low polynomial degree (achievement of node, arc, bound, singleton consistencies).

W-AC* W-AC* is a recent notion of consistency in WCSPs based on an inherent cost of the problem, $C_\emptyset$, common to any solution. It also employs $C_{x_i}[v]$, an inherent cost of each value v for each variable x_i. $C_{x_i}[v]$ is an additional cost (besides $C_\emptyset$) appearing in any solution where v is assigned to x_i. It is defined based on an upper bound B for the cost of an acceptable solution. The initial definition of W-AC* considered only binary constraints, but its extension to n-ary constraints is straightforward and we present it directly.

Definition 3 (NC*). *Let $P = (X, D, C)$ be a WCSP with B the lowest known upper bound for an acceptable solution. $\langle x_i, v \rangle$ is node consistent if $C_\emptyset + C_{x_i}[v] < B$. Variable x_i is node consistent if: i) all its values are node consistent and ii) there exists a value $v \in D_i$ such that $C_{x_i}[v] = 0$. Value v is a support for variable's x_i node consistency. P is node consistent (NC*) if every variable is node consistent.*

Definition 4 (AC*). *$\langle x_i, v \rangle$ is arc consistent with respect to constraint ϕ_k if it is NC* node consistent and there is a tuple of assignments ϵ for all variables in X_k such that $\epsilon|_{\{x_i\}} = \{\langle x_i, v \rangle\}$ and $\phi_k(\epsilon) = 0$. Tuple ϵ is called a support of v. Variable x_i is arc consistent if all its values are arc consistent with respect to every constraint affecting x_i. A WCSP is arc consistent (AC) if every variable is arc consistent.*

Initially $C_x[v]$ and $C_\emptyset$ are set to 0. For n-ary WCSPs, W-AC* runs by iteratively increasing different $C_x[v]$ by projections from some constraint ϕ_k, or decreasing a $C_x[v]$ by projecting it unto $C_\emptyset$, until a fix point is achieved. Namely, the fix point may depend on the order of the operations. $C_x[v]$ is increased by subtracting and transferring to it $\min_{\{\epsilon | \langle x, v \rangle \in \epsilon\}} \phi_k(\epsilon|_{X_k})$, from all the values $\{\phi_k(\epsilon|_{X_k}) | \langle x, v \rangle \in \epsilon\}$, enforcing AC arc consistency. Each value v such that $C_x[v]+C_\emptyset$ is higher than the current bound for acceptable solution is removed, achieving NC consistency. A common share of $\min_v C_x[v]$ of each weight accumulated in $C_x[v]$ is transferred to $C_\emptyset$, and this last type of operation obtains what is called NC* node consistency.

3 Asynchronous algorithms

In solving distributed problems, needs of synchronization make the whole system run at the speed of the slowest link. Synchronizations also force agents to communicate all the private information due in that round (usually called epoch), even if some divulgations can be proven unnecessary by subsequent communications.

A solution is to let the distributed solving run *asynchronously*. Namely the agents need not synchronize but rather they can flexibly exchange messages of several types. Only a subset of these messages are required in order to guarantee correctness properties, but even those messages can be sent with a bounded delay. As explained before, this improves robustness to slow links and handling of privacy. [13, 1].

We propose an asynchronous algorithm that exploits consistency maintainance concurrently with search in a way that lets the most promising behavior to emerge. It consist in running asynchronous backtracking (ABT), on top of which distributed 'local' consistency achievement is enforced independently and concurrently on several subproblems that are defined dynamically.

ABT ABT is a technique currently known for solving DisCSPs, where each agent A_i asynchronously and concurrently with the other agents performs Backtracking's actions to assign x_i with a value from D_i [16]. The assignment is announced via **ok?** messages to lower priority agents, namely agents A_j, $j{>}i$. In making her assignment A_i must satisfy ϕ_i given the other assignments she knows for X_i from higher priority agents. When this task is impossible, A_i announces the lowest priority agent A_k among those whose assignments explain the conflict, by sending her the explanation. A conflict explanation is called *nogood* and consist of the set of inconsistent assignments. This explanation is integrated in ϕ_k. A_i also removes that assignment of x_k, which should allow her to now make her assignment or find a second conflict (recursively calling **backtrack**).

A further improvement consists of allowing agents to send new assignments and labels for a variable x_i only to those agents A_j that have constraints involving x_i. We say that A_j is interested in x_i. When a new agent receives a conflict explanation creating a constraint on a new variable x_k, it announces this to A_k via an **add-link** message.

Algorithms 1 and 2 contain the pseudocode performing the functionalities described above. **check_agent_view** is where agents try to make their assignment and **backtrack** is where they treat conflicts as described above. Each agent A_i maintains a set *outgoing-links* naming the other agents having constraints on the variable it owns, x_i. It also maintains an *agent-view*, namely the set of latest proposals received from each agent and that are still *valid* (i.e. they are not yet expected to have been changed). Agents exchange **ok?**, **add-link**, and **nogood** messages with instantiation proposals, interests, respectively nogoods:

- **ok?** messages are sent from A_i to lower priority agents proposing a value v_i for x_i (timestamped with c_{x_i}).

- **add-link** messages are sent from an agent A_i to a higher priority agent A_j to show interest in current and future proposals for x_j. It is normally due to the acquisition of constraints on x_j by A_i.

- **nogood** messages are sent from an agent A_i to a higher priority agent A_j to announce an impossible partial valuation of even higher priority proposals due to the value of x_j it

when received *(***ok?**,$\langle x_j, v_j, , c_{x_j}\rangle$*)* **do**
  if(old c_{x_j}) **then** return;
  ***if**(j≤cL$_i$) **then** cL$_i$←i*//only in ABT-cL;
  add(x_j,v_j,c_{x_j}) to *agent view*;
  eliminate invalidated nogoods;
  ***maintain_consistency**(j)*//only in DisWMAC;
  check_agent_view; //only satisfies consistency nogoods of levels t, t<cL$_i$, in ABT-cL;
end do.
procedure check_agent_view do
  when agent view *and* current_value *are not consistent //cf. nogoods of levels t, t<cL$_i$*
    if *no value in D_i is consistent with* agent view **then**
      backtrack;
    else
      select $d \in D_i$ where *agent view* and d are consistent;
      current_value ← v; $C^i_{x_i}$++;
      send (**ok?**,$\langle x_i, v, C^i_{x_i}\rangle$) to lower priority agents in *outgoing links*;
    end
end do.

Algorithm 1: Procedures of A_i for receiving **ok?** messages in ABT, ABT-cL, and DisWMAC.

knows.

4 Preliminary adjustments

First modification: ABT-cL In ABT, when a nogood is sent to A_j, x_j is expected to change and is therefore removed from *agent_view*. In typical versions with polynomial space requirements, forgetting assignments leads to forgetting the nogoods in which they are involved.

We do not want to forget data structures (namely consistency labels) that may be stored by other agents. This can lead to persistent inconsistency in the views of the agents and a distributed W-AC* algorithm may not converge to a consistent fix point.

For integration with the proposed algorithm, a threshold cL$_i$ is introduced for each agent A_i (initially having value i). An agent checks the consistency of an assignment only versus assignments received from agents A_j, $j<cL_i$. When a nogood is sent to A_k, cL$_i$ is reduced to k. The agent can try again to make her assignment. cL$_i$ is reset to i when an **ok?** message is received from A_k, k≤cL$_i$. This version is called ABT-cL.

Lemma 1. *After each sending of a nogood message (which reduces the cL$_i$), either A_i will eventually receive an assignment for a variable x_j, j≤cL$_i$ (which resets cL$_i$ to i), or failure is detected.*

Proof The agent A_{cL_i} will either change its assignment, or the conflict specified in the nogood is invalidated by some previous change of assignment, or recursively A_{cL_i} sends a nogood for which it expects a new assignment.

The recursion either goes until an empty nogood is detected (e.g. at latest by the first agent), and failure is detected, or changes of assignments are found to invalidate all the conflicts specified in the generated nogoods (to a variable $x_j, j<cL_i$ from the nogood). q.e.d.

Theorem 1. *ABT-cL is correct, complete and terminates.*

when received *(***nogood**,A_j,$\neg N$*)* **do**
  if *((($\langle x_i, v, c\rangle \in N \wedge (A_i$ knows $(M \rightarrow (x_i \neq v))) \wedge \neg$(better $\neg$N than $\neg$M)) $\vee$ invalid($\neg N$)))*
  then
    if *(current version stores all nogoods)* **then**
      when $\langle x_k, v_k, t_k\rangle$*, where* x_k *is not connected, is contained in* $\neg N$
        send **add-link** to A_k;
        add $\langle x_k, v_k, t_k\rangle$ to *agent_view*;
      store $\neg N$;
    end
  else
    when $\langle x_k, v_k, t_k\rangle$*, where* x_k *is not connected, is contained in* $\neg N$
      send **add-link** to A_k;
      add $\langle x_k, v_k, t_k\rangle$ to *agent_view*;
    put $\neg N$ in *nogood-list* for x_i=v;
    add all new assignments in $\neg N$ to *agent_view*;
2.1 reconsider stored and invalidated nogoods;
  end
  __maintain_consistency__(smallest modified level)//only DisWMAC;
  check_agent_view;
end do.
procedure backtrack do
  nogoods $\leftarrow \{V | V=$inconsistent subset of *agent view*$\leq$ cL$_i\}$;
  when *an empty set is an element of* nogoods
    broadcast to other agents that there is no solution; terminate this algorithm;

  for every $V \in$ nogoods **do**
    select $\langle x_j, v_j, t_{x_j}\rangle$ where x_j has the lowest priority in V;
2.2 send (**nogood**,A_i,V) to A_j;
    cL$_i$$\leftarrow$j$-$1 // only ABT-cL$_i$;
  end do
  check_agent_view;
end do.

Algorithm 2: Procedures of A_i for receiving **nogood** messages in ABT, ABT-cL, and DisWMAC.

Proof *Correctness (i.e. at quiescence the state is a solution)*: From Lemma 1 it results that at quiescence (without detecting failure) all cL_i are equal with i. Also, at quiescence all agents know the last assignments of their predecessors based on timestamps. Therefore each agent A_i is consistent with all assignments to previous variables having ϕ_i satisfied. This means that the set of all assignments satisfies all constraints.
Completeness (i.e. failure cannot be detected if there is a solution): All used nogoods are generated based on logical inference. No failure can therefore be inferred if a solution exists.
Termination: Recursively for i growing from 1 to n, once agents $A_j, j<i$ no longer change their assignments, A_i either exhausts its domain generating a valid nogood leading to failure in finite time, or one of its proposals will never be refused with a nogood. q.e.d.

The technique proposed in ABT-cL is somewhat similar to a mechanism we used in [11], but here the reseting of cL$_i$ to i on the receipt of an **ok?** message had to be made explicit.

when received *(***solution***,B)* **do**

add $\phi(x)$ to the set of local constraints:

$$\phi(x) = \begin{cases} \infty & \text{if } \sum_{\text{known } x_{c_i}} x_{c_i} \geq B \\ 0 & \text{if } \sum_{\text{known } x_{c_i}} x_{c_i} < B \end{cases}$$

end do.

procedure solution-detected *(solution)* **do**

$B \leftarrow \sum_{(x_{c_i}, C_i, k_i) \in solution} C_i$;

broadcast (**solution**,B)

end do.

Algorithm 3: Procedure of A_i for receiving **solution** messages in ABT-cL-BB. All other procedures are inherited from ABT-cL. The procedure *solution-detected* is run by whoever detects and builds the solution. If each agent builds the solution separately then the message needs not be broadcast but just delivered locally.

Second modification: Branch and Bound Soft (weighted) constraints alone cannot be used to prune the search space. The extension of ABT-cL to DisWCSPs is based on exploiting the hard constraints (bounds) defined by already found solutions. Let us introduce a new variable x_{c_i}, $x_{c_i} \geq 0$ for each agent A_i. These variables model the cost of the current proposal, i.e. the value of ϕ_i. Since all agents are interested in the variables x_{c_i}, all the agents are in the *outgoing-links* of each agent A_i for the variable x_{c_i}. A_i proposes $x_{c_i}=k$ when his proposal for x_i has cost of local constraints k.

Example 2. *E.g. consider A_2 having ϕ_2 given in Figure 1, and having in her agent_view $x_1 = 0$. x_{c_2} is assigned k=4 when the proposal of A_2 is $x_2 = 2$.*

If her agent_view is empty, A_2 picks a cost of her choice among those possible for her proposal. E.g. when proposing $x_2 = 2$ without knowing x_1, A_2 could pick $x_{c_2} = 2$ (which is otherwise true only if $x_1 = 3$).

In Branch & Bound (B & B) the idea is to discard search paths for which it is proven that any enclosed solution is more expensive than some already found solution. Any solution with value B defines therefore a *nogood* (i.e. dynamically inferred constraint), $\sum_i x_{c_i} < B$, that is broadcast to all agents. It is known that $x_{c_i} \geq 0$, therefore each agent can enforce the weaker constraint:

$$\sum_{\text{known } x_{c_i}} x_{c_i} < B$$

No other modification is required and a new B & B algorithm is obtained. The last found solution is optimal. This algorithm is called ABT-cL-BB.

Remark 1. *With ABT-cL-BB, the value of a solution is given by the sum of the values assigned in it to the x_{c_i} variables.*

Each time that a solution is detected, a **solution** message is broadcast to participants, having as parameter the value B of the obtained solution. Algorithm 3 shows how the constraint $\sum_i x_{c_i} < B$ is added to each agent. Initially $B = \infty$ and agents enforce $\sum_i x_{c_i} < \infty$.

Proposition 2. *ABT-cL-BB is correct, complet, terminates, and finds the optimal solution.*

Proof. *The proof is immediate from the correctness of ABT-cL and by construction (introduction of B & B behavior which is known to be correct).*

It was already mentioned that Branch & Bound can be added to several asynchronous techniques [12]. There is no specific difference in the way in which it was added here to ABT-cL.

5 Asynchronous maintainance of W-AC*

We propose to maintain arc consistency concurrently for each subproblem P_k, $k \in \{0..n\}$, generated by adding to DisWCSP the last proposed assignments of agents $A_i, i \leq k$. Intuitively, this is done by having each agent $A_j, j > k$ asynchronously and concurrently compute consistent labels for her view of the problem P_k. Her view of a DisWCSP consists of ϕ_j and of the labels it knows. The labels modified by this computation are sent by A_j to all agents $A_i, i \geq k$.

5.1 Factoring out weights

In the previous section it can be noticed that in ABT-cL-BB, cost conflicts were only detected from partial valuations. A better idea has been introduced for centralized techniques in [5, 6, 9], where propagation of labels also estimate costs. The most promising among them is W-AC*. We propose to exchange W-AC* labels, C_x and $C_\emptyset$, obtained by A_i together with the assignments explaining them, under the form of a Weighted Consistency Label which has a stand-alone logic semantic.

Definition 5 (Weighted Consistency Label). *A weighted consistency label (WCL) for a level (i.e. search depth) k and a variable x has the form $\langle A_i, x, k, C^i_x[k], C^i_\emptyset[k], V \rangle$.*

V is a set of assignments. Any assignment in V must have been proposed by A_k or its predecessors. A_i is the agent computing the WCL. C^i_x is a set of inherent additional costs of each value of x given V and when the cost inherent to the problem of A_i is $C^i_\emptyset[k]$.

An assignment is valid if no assignment with a newer timestamp is known for the same variable.

Definition 6 (valid weighted nogood). *A WCL is valid only as long as all the assignments involved in it are valid.*

5.2 Considerations in chosing Data Structures for DisWMAC

The family of algorithms proposed here, DisWMAC, builds on ABT-cL-BB by adding the use of WCLs for propagating costs.

The following approaches are known for maintaining data structures with nogood-based consistency (considering that labels are treated as ranges):

- DMAC0: Storing only the last valid consistency nogood (CN) per variable (related to what was done in [3] for each value).
- DMAC1: Storing only the last valid CN per variable per search depth (as in MHDC [12]).
- DMAC2: Storing only the last valid CN per variable per search depth per agent generating CNs (as in the version of DMAC proposed in [11]).

- DMAC3: Storing only the last valid CN per variable per search depth per agent generating CNs and per agent whose constraints are not involved in the CN (as in [12] for robustness in treating openness).

All of the previous four alternatives translate to DisWMAC, storing WCLs instead of CNs.

We recall that maintainance of W-AC* is done by modifying constraints. Therefore the agents also need to store a distinct copy of her constraint for each consistency level, tagged with the view based on which it was modified. The resulting techniques are therefore called: DisWMAC0, DisWMAC1, DisWMAC2, DisWMAC3.

In the previous versions each agent reinforces separately the consistency of the labels. With W-AC*, the problem common cost $C_\emptyset$ cannot be reliable computed locally if an agent does not know all the labels of all variables. This requires that every WCL is sent to all agents in that level.

A way to alleviate this requirement is by deciding agents responsible for the fraction of $C_\emptyset$ coming out of each variable. Namely, A_i will be responsible for computing and distributing the increment of $C_\emptyset$ that is obtained from C_{x_i}. In the obtained scheme, DisWMAC4, each agent A_i needs to store for each level k, and agent A_j:

- the last WCL received from A_j for the variable x_i at level k, including the last $C^j_\emptyset[k]$ received from each A_j,
- and a WCL for each x_j from agent A_j.

The advantage of DisWMAC4 is that A_i will not need to announce everybody each change in each set $C^i_x[k]$. She only needs to announce changes of $C^i_{x_j}[k]$ to A_j. Changes to $C^i_\emptyset[k]$ still have to be announced to all agents $A_j, j{\geq}k$. A_i is the single agent that can know all the values that can be safely removed from the domain of x_i and on each modification of the label of x_i at level k it has to send a WCL to each other agent $A_j, j{\geq}k$.

5.3 The DisWMAC4 Algorithm

The DisWMAC4 algorithm illustrates well the tradeoffs in maintaining W-AC*. As shown before, W-AC*, maintains an inherent cost of the problem, $C_\emptyset$, that will be in any solution. It also maintains an incremental inherent cost of each value v for each variable x, $C_x[v]$. $C_x[v]$ occurs in any solution where x is assigned to v. In DisWMAC4, A_i is responsible for computing and distributing the increment of $C_\emptyset$ that can be obtained from C_{x_i}. The structures used in Algorithm 4 are:

- B is the cost of best solution announce so far.
- $c^k_{x_v}(j,i)$ is the last timestamp received by A_i from A_j for a WCL for x_v at level k. In DisWMAC4 v is either j or i.
- $\phi_i[k]$, is the vector of weighted constraints of A_i at each level k together with an explaining view.
- $K^i_{x_j}[k]$ is the inherent cost vector for each value of variable x_j as inferred from the constraints of A_i at level k. It is computed by A_i and sent only to A_j together with an explaining view.

when received*(***propagate***,* A_j,x_v,k,$C'^{j}_{x_v}[k]$,$C^{j}_{\emptyset}[k]$,$c^{k}_{x_v}(j)$*, V)* **do**
3.1 **when** have higher tag $c^{k}_{x_v}(j,i) \geq c^{k}_{x_v}(j)$ **then** return;
$c^{k}_{x_v}(j,i) \leftarrow c^{k}_{x_v}(j)$;
when any $\langle x, v, c\rangle$ in V is invalid (old c) **then** return;
when $\langle x_u, v_u, c_u\rangle$*, with a not connected* x_u *is in V*
send **add-link** to A_u;
add $\langle x_u, v_u, c_u\rangle$ to *agent view*;
3.2 add other new assignments in V to *agent view*;
eliminate invalidated nogoods and structures;
if (i=v) store $C'^{j}_{x_v}[k]$ as $K^{j}_{x_i}[k]$, **else** as $C^{j}_{x_v}[k]$;
also store $C^{j}_{\emptyset}[k]$, both tagged by V;
maintain_consistency(min level that is modified);
check_agent_view; //up to level t, t≤cL$_i$;
end do.
procedure maintain_consistency*(minT)* **do**
if (minT $>$ cL$_i$) **then** return;
3.3 **for** *(t←minT; t≤i; t++)* **do**
new-cns ← local-W-AC*(t);
when *(domain wipe out explained by* nogoods*)*
if *finding an empty nogood* **then**
broadcast failure;
end
for every $V \in$ nogoods **do**
select $\langle x_j, v_j, c_{x_j}\rangle$ where x_j has the lowest priority in V;
3.4 send (**nogood**,A_i,V) to A_j;
cL$_i$ ←min(j,cL$_i$);
end do
break;
for every *(*$K^{i}_{x_u}[t]$,$V^{i}_{u}[t]$*)* ∈ new-cns **do**
when $C^{i}_{x_u}[t]$ *is modified*
$c^{t}_{x_u}(i)$++;
3.5 send (**propagate**,A_i,x_u,t,$K^{i}_{x_u}[t]$,$C^{i}_{\emptyset}[t]$, $c^{t}_{x_u}$, $V^{i}_{u}[t]$) to A_u;
end do
when $C^{i}_{x_i}[t]$ *is modified*
3.6 send (**propagate**,A_i,x_i,t,$C^{i}_{x_i}[t]$,$C^{i}_{\emptyset}[t]$,$c^{t}_{x_i}$,$V^{i}_{i}[t]$) to all other agents $A_j, j \geq t$ interested in x_i;
when $C^{i}_{\emptyset}[t]$ *is modified*
3.7 send (**propagate**,A_i,x_i,t,$C^{i}_{x_i}[t]$,$C^{i}_{\emptyset}[t]$,$c^{t}_{x_i}$,$V^{i}_{i}[t]$) to all other agents $A_j, j \geq t$;
end
end do.

Algorithm 4: Processing received WCLs.

- $C^{j}_{\emptyset}[k]$ is the inherent cost due to the variable x_i at level k. It is computed and distributed by A_j together with an explaining view.
- $C^{i}_{x_j}[k]$ is the currently accumulated cost of all agents for assigning x_j to any of its values

```
procedure local-W-AC* (t) do
    update C^i_{x_i}[t] and C^i_∅[t]; C_∅ ← Σ_j C^j_∅[t];
    until convergence;
4.1 for every variable x_v achieve NC for x_v in φ_i[t];
    for every active value u of x_v do
        if value u in x_v has no support on a variable than create support for x_v=u, transfering
        φ_i[t] tuple weights unto K^i_{x_v}[t][u];
        K^i_{x_i}[t][u] propagates to C^i_{x_i}[t][u];
4.2     enforce NC* of x_i by reducing C^i_{x_i}[t][u] unto C^i_∅;
4.3     reinforce NC on x_v considering weight increments
                                                  of K^i_{x_v}[k];
    end do
    return set of modified pairs (K^i_{x_u}[k],V^i_u[k]);
end do.
```

Algorithm 5: Local W-AC* computation.

in the current search branch at level k. It is summed up and distributed by A_j together with an explaining view.

- $C_\emptyset[k]$, is the global cost used during local W-AC* computations at level k.

Each agent A_i locally enforces W-AC* at each level k, where $C_\emptyset$ is given by the sum between all received $C^j_\emptyset[k]$ and a $C^i_\emptyset[k]$ that can be extracted from summing learned vectors $K^j_{x_i}[k]$ for each j. For this each agent computes the sum of all $K^j_{x_i}[k]$ that she learns and decomposes it into a $C^i_\emptyset[k]$ and a $C^i_{x_i}[k]$ by enforcing NC*.

$C^i_\emptyset[k]$ and $C^i_{x_i}[k]$ must be recomputed on each relevant change. Each agent A_i computes its $C^i_\emptyset[k]$ by applying NC* only on x_i. All other variables are made consistent with NC. Also, each agent A_i sends with WCLs for x_i, $C^i_{x_i}[k]$, while for other variables x_j it sends $K^i_{x_j}[k]$. The improvement suggested at line 4.3 allows to avoid need of communication for propagating locally prunnings of a foreign variable detected by an agent. WCNs are exchanged via a new type of messages, **propagate**, and a pseudocode of the described technique is proposed in Algorithm 4.

Remark 2. *When the value transfered by AC from the weights of a constraint $\phi_i[k]$ to $C^i_x[v]$ is ∞, then all the weights of $\phi_i[k]$ for $x=v$ can be set to 0 (this changes nothing as they are anyhow removed by NC).*

Lemma 2. *Distributed W-AC* for a subproblem defined by a given set of assignments V converges in finite time.*

Proof Distributed W-AC* only sends a **propagate** message if:

- An infinite weight of ϕ_i is reduced to 0, to increase an element of a vector C^i_x (see Remark 2).

- A non-infinite weight of ϕ_i is reduced to increase an element of a vector C^i_x.

- If weights of C^i_x are transfered to $C^i_\emptyset$.

Since the sum of non-infinite weights of each ϕ_i is finite, the total number of such operations that is possible is finite, therefore the distributed W-AC* terminates in finite time. q.e.d.

Theorem 3. *DisWMAC4 is correct, complet and terminates.*

Proof *Correctness (at quiescence without detecting failure the valuation is a solution)*: As in Theorem 1, from Lemma 1 it results that if no failure is detected then each agent is consistent with predecessors and all constraints are satisfied at quiescence.
Completeness (no failure can be detected if a solution exists): All value removals and needs of assignment changes in each subproblem are based on logical inference. Therefore, if a solution exists no failure can be inferred.
Termination: Similarly to Theorem 1, recursively it can be shown that in finite time after agents $A_j, j<i$ no longer change their assignments, A_i will stop receiving propagate messages as the distributed W-AC* is known to converge. Then either A_i generates a valid nogood leading to detection of failure, or one of its proposals will never be rejected with nogoods. The removal of additional values from domains can only speed up termination. q.e.d.

Example 3. *Two possible runs of DisWMAC4 for a DisWCSP with two agents A_1 and A_2 and a single constraint (between x_1 and x_2) enforced by A_2 is shown in Figure 2. The constraint of A_2 is the one in Figure 1. The traces differ by the order in which local-W-AC* enforces AC on its variables. Here we do not detail the way in which an agent S can detect the solution, as several such techniques are well known. The first found solution has cost 2, and after this bound is added as a constraint by everybody, the next found solution has cost 1. When the bound 1 is added, A_2 can immediately detect that there is no better solution than this.*

6 Discussion

DisWMAC3 The DisWMAC3 algorithm needs slightly more structures as each agent must store all $K_x^j[k]$ of each other agent. It also requires that each change of $K_{x_j}^i[k]$ be sent to all other agents $A_t, t\geq k$, while DisWMAC4 does so only for changes in $C_\emptyset^i[k]$ (see lines 3.7 and 3.6). Nevertheless, DisWMAC3 is simpler, not having to communicate any $C_\emptyset^i[k]$, as each agent detects them locally. Moreover, the communication delay can be reduced by half due to the fact that no intemediary agent stays in the middle of a communication.

A* Recently, another technique, called Adopt [8], shows how A* value ordering heuristic can be introduced in ABT. The Maintaining Consistency technique proposed here acts in DisWMAC as a heuristic estimator in each node. When the algorithm works in A* mode, namely abandoning each node when its heuristic value is higher than the value of another alternative, then the theory of A* applies. In this case, a dominant heuristic expands a strictly smaller search space.

Definition 7. *In asynchronous A* search, a heuristic function h_1 is dominant over h_2 if at anytime, the value it estimates is higher or equal to h_2, and still admissible (optimistic).*

Using a powerful technique for estimating the heuristic function is dominant only as far as it performs constantly better. In practice it is seldom that two search techniques can be compared in this way. Experimentation is therefore required to verify which one works better, but one expects that a technique that in general is better in search, is also better as a heuristic.

First scenario:

1: A_1	**ok?**$\langle x_1, 0, 1\rangle\langle x_{c_1}, 0, 1\rangle$	A_2
2: A_2	**propagate**(A_2,x_1,0,(2,2,1,1),0,$\emptyset$)	A_1
3: A_2	**propagate**(A_2,x_2,0,(0,1,1,2),0,$\emptyset$)	A_1
4: A_2	**propagate**(A_2,x_1,1,(_,2,_,_),0,$\langle x_1, 0, 1\rangle$)	A_1
5: A_2	**propagate**(A_2,x_2,1,(0,2,3,2),0,$\langle x_1, 0, 1\rangle$)	A_1
6: S	**solution**(2)	everybody
7: A_2	**nogood**($\langle x_1, 0, 1\rangle$)	A_1
8: A_1	**ok?**$\langle x_1, 1, 2\rangle\langle x_{c_1}, 0, 2\rangle$	A_2
9: A_2	**nogood**($\langle x_1, 1, 2\rangle$)	A_1
10: A_1	**ok?**$\langle x_1, 2, 3\rangle\langle x_{c_1}, 0, 3\rangle$	A_2
11: A_2	**propagate**(A_2,x_1,1,(_,_,1,_),0,$\langle x_1, 2, 3\rangle$)	A_1
12: A_2	**propagate**(A_2,x_2,1,(0,4,4,6),0,$\langle x_1, 2, 3\rangle$)	A_1
13: S	**solution**(1)	everybody
14: A_2	**nogood**(fail)	everybody

Second scenario:

1: A_1	**ok?**$\langle x_1, 0, 1\rangle\langle x_{c_1}, 0, 1\rangle$	A_2
2: A_2	**propagate**(A_2,x_2,0,(0,1,1,3),1,$\emptyset$)	A_1
3: A_2	**propagate**(A_2,x_1,0,(1,1,0,0),1,$\emptyset$)	A_1
4: A_2	**propagate**(A_2,x_2,1,(1,4,3,2),2,$\langle x_1, 0, 1\rangle$)	A_1
5: A_2	**propagate**(A_2,x_1,1,(_,0,_,_),2,$\langle x_1, 0, 1\rangle$)	A_1
6: S	**solution**(2)	everybody
7: A_2	**nogood**($\langle x_1, 0, 1\rangle$)	A_1
8: A_1	**ok?**$\langle x_1, 1, 2\rangle\langle x_{c_1}, 0, 2\rangle$	A_2
9: A_2	**nogood**($\langle x_1, 1, 2\rangle$)	A_1
10: A_1	**ok?**$\langle x_1, 2, 3\rangle\langle x_{c_1}, 0, 3\rangle$	A_2
11: A_2	**propagate**(A_2,x_2,1,(0,4,4,6),1,$\langle x_1, 2, 3\rangle$)	A_1
12: A_2	**propagate**(A_2,x_1,1,(_,_,0,_),1,$\langle x_1, 2, 3\rangle$)	A_1
13: S	**solution**(1)	everybody
14: A_2	**nogood**(fail)	everybody

Figure 2: For different orders of variables in W-AC*, two traces of DisWMAC4 when the constraint in the first example is the only one in the problem with two agents, A_1 and A_2, where the solution is detected by some agent S.

7 Conclusions

Distributed optimization is an expensive task. Most approaches use hill-climbing and approximate techniques [7]. Several other synchronous Branch & Bound and A* techniques appeared last decade mainly in work reported by Dr. Yokoo's team. Preliminary tests that we performed on ABT/AAS based B & B have shown the technique to be prohibitively expensive on a simple real-world problem. While it is not yet known how the two existing asynchronous optimization techniques compare (namely asynchronous A* vs asynchronous Branch & Bound), here we have shown a powerful heuristic that can be used with both of them. In as much as consistency maintenance dominates forward checking, the new heuristic promises to dominate the ones used so far.

The main new ideas proposed in this article are that:

1. Consistency achievement or maintenance in Weighted DisCSPs can be performed if ABT-cL-BB is enriched to a more general concept: The Weighted Consistency Label (WCL).

2. An asynchronous equivalent of the best available centralized technique, B & B with W-

AC*, is obtained by mixing the aforementioned consistency maintenance with Branch & Bound.

3. The feedback that A* needs about low bounds on constraints of successor agents, can be extracted using cost attached to labels in WCLs and detected by the previously mentioned 'local' consistency process.

In this paper we outlined the steps required for *asynchronizing* B & B with W-AC for Distributed Weighted CSPs.

References

[1] J. Denzinger, M. Silaghi, and M. Yokoo. Distributed constraint reasoning. In *IJCAI Tutorial SP3*, 2003.

[2] Boi Faltings. Incentive compatible open constraint optimization. In *Electronic Commerce*, 2003.

[3] Narendra Jussien, Romuald Debruyne, and Patrice Boizumault. Maintaining arc-consistency within dynamic backtracking. In *CP'2000*, Singapore, 2000. Springer.

[4] J. Larrosa. Node and arc consistency in weighted csp. In *AAAI-2002*, Edmonton, 2002.

[5] Javier Larrosa, Pedro Meseguer, and Thomas Schiex. Maintaining reversible DAC for Max-CSP. *AI*, 107:149–163, 1999.

[6] Javier Bondia Larrosa. *Algorithms and Heuristics for Total and Partial Constraint Satisfaction*. PhD thesis, IIIA,Bellaterra,Spain, 1998.

[7] Michel Lemaître and Gérard Verfaillie. An incomplete method for solving distributed valued constraint satisfaction problems, 1997.

[8] Pragnesh Jay Modi, Milind Tambe, Wei-Min Shen, and Makoto Yokoo. A general-purpose asynchronous algorithm for distributed constraint optimization. In *Distributed Constraint Reasoning, Proc. of the AAMAS'02 Workshop*, Bologna, July 2002. AAMAS.

[9] T. Petit, J.C. Régin, and Bessière C. Range-based algorithm for max-csp. In *Proceedings of CP*, pages 280–294, Ithaca, September 2002.

[10] S. Russell and P. Norvig. *Artificial Intelligence, A Modern Approach.* Prentice Hall, 2nd edition, 2002.

[11] M.-C. Silaghi, D. Sam-Haroud, and B.V. Faltings. Consistency maintenance for ABT. In *Proc. of CP'2001*, pages 271–285, Paphos,Cyprus, 2001.

[12] Marius-Călin Silaghi. *Asynchronously Solving Distributed Problems with Privacy Requirements*. PhD Thesis 2601, (EPFL), June 27, 2002. http://www.cs.fit.edu/~msilaghi/teza.

[13] Marius-Călin Silaghi and Boi Faltings. Openness in asynchronous constraint satisfaction algorithms. In *3rd DCR-02 Workshop*, pages 156–166, Bologna, July 2002.

[14] R. Wallace and M.C. Silaghi. Using privacy loss to guide decisions in distributed CSP search. In *FLAIRS'04*, 2004.

[15] R.J. Wallace and E.C. Freuder. Constraint-based multi-agent meeting scheduling: Effects of agent heterogeneity on performance and privacy loss. In *DCR*, pages 176–182, 2002.

[16] M. Yokoo, E. H. Durfee, T. Ishida, and K. Kuwabara. The distributed constraint satisfaction problem: Formalization and algorithms. *IEEE TKDE*, 10(5):673–685, 1998.