# Improving Asynchronous Partial Overlay

Roger Mailler
*Tandy School of Computer Science*
*University of Tulsa*
*Tulsa, OK*
*Email: mailler@utulsa.edu*

***Abstract*—Since its creation, the Asynchronous Partial Overlay (APO) protocol has received a great deal of attention because of its non-traditional approach to solving Distributed Constraint Satisfaction Problems (DCSPs). Its introduction led investigators to question the very definition of the word "distributed" and has subsequently inspired the community to create improved metrics for parallel computation, enhanced testing procedures, and most importantly new DCSP algorithms. These advances have raised concerns about APO's parallel efficiency by showing that, in some cases, APO performs very poorly compared to protocols such as Asynchronous Forward Checking, Conflict-directed Backjumping (AFC-CBJ). In addition, APO's soundness and completeness were brought into question when it was discovered that, under certain conditions, stale state information could cause the protocol's distributed locking mechanism to fail.**

**This work revisits APO by re-engineering the protocol to simplify it and increase its parallelism while ensuring its soundness and completeness. It also vastly improves the parallel efficiency of APO by replacing its central solver with a variant of the Forward Checking, Conflict-directed Backjumping (FC-CBJ) algorithm that is specifically tuned to complement the heuristic strategies used by APO to limit its centralization. This new version of APO is then evaluated against the AFC-CBJ protocol using random instances of both DCSPs and distributed 3-coloring problems. The end result is a protocol that is several orders of magnitude faster than the original APO, uses less messages, is more private, and outperforms the AFC-CBJ protocol in nearly every case tested.**

***Keywords*-Constraint Satisfaction; Partial Centralization; Distributed Search;**

## I. Introduction

The Asynchronous Partial Overlay (APO) protocol [1] is unlike any other sound and complete protocol for solving Distributed Constraint Satisfaction Problems (DCSPs). The fundamental difference lies in its method of using structural cues, obtained through feedback during the search process, to perform controlled centralization. The protocol can therefore be described as using dynamic, partial centralization as its problem solving technique.

At the time of its introduction, APO was shown to be a very efficient protocol for solving relatively large 3-coloring problems and was believed to be sound and complete [2]. However, the original comparison between APO and Asynchronous Weak Commitment (AWC) [3] has been questioned because the metrics used for the evaluation did not account for the cost of executing the centralized solver [4]. One performance measure that has gained popularity to resolve this issues is to count the number of non-concurrent constraint checks (NCCCs) performed during the problem solving process [5]. Using this metric, several studies have shown that the APO protocol performs quite poorly compared to AWC and the Asynchronous Forward Checking, Conflict-directed Backjumping (AFC-CBJ) [6] protocols, particularly in the phase transition of problems with large domains [7].

In addition to these findings, it was also discovered that stale information and partial mediation sessions could lead to oscillations in the protocol [7]. These concerns, although easily repaired, raised serious doubts about the validity of the protocol that still linger today.

This work uses an expanded understanding about the nature of APO to re-engineer the information exchange and control mechanisms used by the protocol during problem solving. The most dramatic changes are the removal of the requirement for bi-directional linking, the relaxation of the need for continuous state updating in non-neighboring agents, the introduction of a preemptive locking mechanism, and finally the elimination of the wait! response to mediation session requests. These changes lead to a reduction in the number of messages used, improved privacy for the agents, and most importantly, simplify the mechanics of the distributed locking used to guarantee that the protocol is sound and complete.

This work also investigates the impact of replacing the original solver in APO with a Forward Checking, Conflict-directed Backjumping (FC-CBJ) [8] constraint solver. This solver is extremely efficient and immediately provides massive reductions in the number of constraint checks performed by APO.

This new version of APO is then evaluated against the AFC-CBJ protocol using random distributed 3-coloring problems, like the original APO studies, and random DCSP problems, like the followup studies. The results of these experiments show that these changes dramatically improve the performance of APO to the point that it outperforms AFC-CBJ on nearly every case and metric tested.

This work is significant because it uses a detailed understanding of the nature of cooperative mediation-based

protocols to reduce the messaging complexity, information exchange, and parallel runtime of APO. We also present the first evaluation of a cooperative mediation-based protocol with a distributed backtracking protocol where the underlying problem solving methods are derived from the same centralized algorithm. In doing so, we also perform the first scalability analysis of AFC-CBJ using distributed 3-coloring. Finally, the most important contribution of this work is that it shows that APO is considerably more efficient than AFC-CBJ.

The rest of this paper is organized as follows. In Section II the DCSP is introduced. In Section III, the original APO protocol is described along with a discussion of its core weaknesses. The section that follows, Section IV, presents the new APO protocol along with its soundness and completeness proofs. In Section V the setup and results of extensive experimental evaluation are given. Finally, in Section VI, we present our conclusion.

## II. Problem Definition

A Distributed Constraint Satisfaction, $P = \langle V, A, D, R \rangle$, consists of the following [9]:

- A set of $n$ variables $V = \{x_1, \ldots, x_n\}$
- A set of $g$ agents $A = \{a_1, \ldots, a_g\}$
- Discrete, finite domains for each of the variables $D = \{D_1, \ldots, D_n\}$
- A set of $m$ constraints $R = \{R_1, \ldots, R_m\}$ where each $R_i(d_{i,1}, \ldots, d_{i,j})$ is function $R_i : D_{i,1} \times \cdots \times D_{i,j} \rightarrow \{true, false\}$. i.e. the constraints take in variable values and return true when the constraint is satisfied or false when it is violated.

The task is to find an assignment $S = \{d_1, \ldots, d_n | d_i \in D_i\}$ where all of the constraints are satisfied or to report that no such solution exists. Each agent is assigned one or more variables along with constraints associated with those variables. The goal of each agent, from a local perspective, is to ensure that the constraints on its variables are satisfied. However, the agents often need to work with one another to ensure that global problem is solved.

This work focuses on the case where each agent is assigned a single variable and the constraints are binary. Since each agent is assigned a single variable, we will refer to the agent by the name of the variable it manages. Because the constraints are binary, we can refer to the graph created by representing variables as vertices and constraints as edges as the *constraint graph*. In addition, two variables are considered to be *neighbors* if they share a constraint. Both of these restrictions can be removed, if needed, either by modifying the algorithms or by converting the non-binary constraints to binary ones [10].

## III. Asynchronous Partial Overlay (APO)

Conceptually, APO is based on the *cooperative mediation* paradigm [1]. Cooperative mediation entails three main principles. The first is that agents use local, centralized search to solve portions of the overall problem. Second, agents use experience to dynamically increase their understanding of their role in the overall problem. Third, agents attempt to promote coherence by both overlapping their knowledge and minimizing their impact on the work done by other agents.

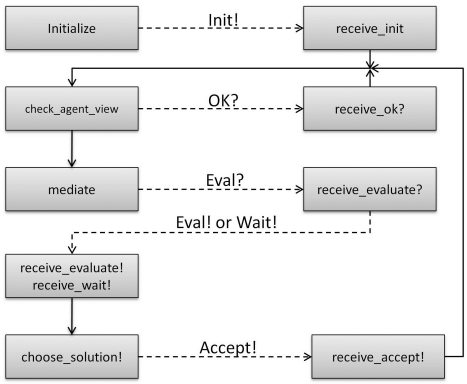


Figure 1. The original APO protocol.

The basic APO protocol is presented in Figure 1. The APO algorithm works by constructing two main data structures; the $good\_list$ and the $agent\_view$. The $agent\_view$ holds the names, values, domains, and constraints of variables to which an agent is linked. The $good\_list$ holds the names of the variables that are known to be connected to the owner by a path in the constraint graph.

As the problem solving unfolds, the agents try to solve the subproblem they have centralized within their $good\_list$ or determine that this subproblem is unsolvable (indicating that the entire problem is overconstrained). To do this, whenever an agent recognizes a constraint violation involving its variable, it takes the role of the mediator and attempts to change the values of the variables within the mediation session to achieve a satisfied subproblem. When this cannot be achieved without causing a violation for agents outside of the session, the mediator links with those agents assuming that they are somehow related to the mediator's variable. This step increases the size of the $good\_list$. This process continues until one of the agents finds an unsatisfiable subproblem, or all of the conflicts have been removed.

### A. Soundness and Completeness of APO

The proofs of APO's soundness and completeness are based on the premise that when a session terminates, there are three possible outcomes. The first is that a subproblem has been identified that has no solution. In this case, the entire problem has been proven to be unsatisfiable and the announcement is made. Second, that the subproblem is satisfiable and has no external conflicts. In this case, the problem has moved one step closer to being conflict free and in situations where the entire problem has been centralized

```
procedure initialize
  d_i ← random d ∈ D_i;
  p_i ← sizeof(neighbors) + 1;
  m_i ← false;
  mediator ← null;
  add x_i to the good_list;
  for ∀x_j ∈ neighbors
    send_init_req (x_j);
end initialize;

procedure send_init_req (x_j)
  send (init?) to x_j;
  add x_j to the good_list;
end;

when received (init?) from x_j
    send (init! x_i, p_i, d_i, m_i, D_i, C_i) to x_j;
end;

when received (init! x_j, p_j, d_j, m_j, D_j, C_j)
  add (x_j, p_j, d_j, m_j, D_j, C_j) to agent_view;
  check_agent_view;
end;

when received (add! x_j)
    send_init_req (x_j);
end;
```

Figure 2. The APO-FCCBJ procedures for initialization and linking.

```
when received (ok? x_j, p_j, d_j, m_j)
  update agent_view with (x_j, p_j, d_j, m_j);
  if x_j ∈ conc_list do
    if d_j generates a conflict with d_i do
      for each (x_j, x_k) ∈ conc_list do
        send(add! x_j) to x_k;
    remove all tuples (x_j, x_k) from conc_list;
  check_agent_view;
end;

procedure check_agent_view
  if sizeof(agent_view) < sizeof(good_list)
     ∨mediate ≠null do
    return;
  m'_i ← hasConflict(x_i);
  if m'_i ∧ ¬∃j(x_j ∈ neighbors ∧ (p_j > p_i)∧
       (hasConflict(x_j) ∨ m_j = true))
    if ∃(d'_i ∈ D_i) (d'_i does not conflict)
      d_i ← d'_i;
      send (ok? x_i, p_i, d_i, m_i) to ∀x_j ∈ neighbors;
    else
      do mediate;
  else if m_i ≠ m'_i
    m_i ← m'_i;
    send (ok? x_i, p_i, d_i, m_i) to ∀x_j ∈ neighbors;
  end if;
end;
```

Figure 3. The procedures for doing local resolution, updating the *agent_view* and the *good_list*.

a globally satisfying solution has been found. Third, that the subproblem is satisfied, but has at least one external conflict. This creates a situation where the agent must expand its context in order to avoid the same situation in the future and allows the protocol, in the limit, to centralize the entire problem ensuring its completeness [1].

The original proofs relied on several complex control mechanisms to ensure that agents always made decisions with the most up-to-date information and that the highest priority mediator could always, *eventually*, obtain a lock on the variables it was mediating over. One of these mechanisms was predictive two-phase locking. This locking mechanism works by having agents continuously exchange **ok?** messages that indicated both their desire to mediate and their current priority value. Using this information agents predict if they are expecting a high priority mediator to request a session and reserve the lock until the request arrives. During this time, the agent can not start a session and tells lower priority mediators to wait. Once a mediator gets the lock, it is not released until the session concludes. This means that even if the highest priority agent requests the lock, it is told to wait. This opened up the possibility of a live-lock where partial mediation sessions occur without the mediators expanding their context [7]. Grinshpoun and Meisels corrected the protocol by adding two new message types that allowed a mediator to **cancel!** sessions when it receives a **wait!** from an agent and to notify mediators to **add!** new links when an inadvertent conflict is created. These additions complicated the soundness and completeness proofs, required extra control logic, limited parallelism, and increased centralization.

### B. *Efficiency of APO*

APO's overall efficiency is closely tied to the centralized solver that the mediator uses to generate solutions. This is true for two reasons. First, the efficiency of the solver direct influences the number of constraint checks that are used to solve a mediator's subproblem. Second, if the solver chooses a solution that is not cohesive with the agents outside of the session then unnecessary linking will occur.

The original version of APO used a backtracking-based branch and bound solver [11] to generate its solutions. According to one of the original publications [1], this solver was chosen because it supports the third principle of cooperative mediation by minimizing the number of external constraint violations it creates, which promotes coherence. However, Mailler and Lesser speculated that improvements in the runtime performance could be obtained by using an improved solver although they have never supported the claim in subsequent work.

Since then, two attempts have been made to replace the centralized solver used in APO. Benisch and Sadeh [4], followed by Grinshpoun and Meisels [7], replaced the branch and bound solver with a simple, backtracking-based CSP solver. Although their solvers were very inefficient, they still reported considerable efficiency improvements on random DCSP instances, but worse performance on random 3-coloring problems. Just as importantly, both also noted considerable increases in the amount of centralization done by the agents. These results reinforce the belief that the poor performance found on random DCSP instances in the original APO was caused by the solver and was easily correctable by using a better search technique. However, the

```
procedure mediate
  preferences ← ∅;
  counter ← sizeof(good_list);
  for each x_j ∈ good_list do
    send (evaluate? x_i, p_i) to x_j;
  mediator ← x_i ;
end;

when received (evaluate! x_j, p_j, lock_j, labeled D_j)
  record (x_j, labeled D_j, lock_j) in preferences;
  update agent_view with (x_j, p_j);
  counter - -;
  if counter = 0 do choose_solution;
end;
```

Figure 4. The APO-FCCBJ procedures for mediating a session.

```
when received (evaluate? x_j, p_j)
  m_j ← true;
  lock ← false;
  if mediator = null do
    mediator ← x_j;
    lock ← true;
  else if mediator ≠ null ∧p_j > p_m do
    mediator ← x_j;
    lock ← true;
  end if;
  label each d ∈ D_i with the names of the agents
    that would be violated by setting d_i ← d;
  send (evaluate! x_i, p_i, lock, labeled D_i);
end;

when received (accept! d, x_j, p_j, m_j, < pairs >)
  for each x_k ∈ neighbors do
    if x_k ∈< pairs > do
      update agent_view with (x_k, d'_k);
    else if d does not conflict d_k do
      add (x_k, x_j) to conc_list;
  if mediator = x_j do
    d_i ← d;
    mediator ← null;
  end do;
  send (ok? x_i, p_i, d_i, m_i) to all x_j ∈ neighbors;
  check_agent_view;
end;
```

Figure 5. APO-FCCBJ procedures for receiving a session.

performance degraded on 3-coloring problems because the new solvers traded local efficiency for incoherence creating additional centralization and ultimately worse overall performance.

## IV. Redesigning APO

The redesign of the APO protocol (see Figures 2 through 6) focuses on removing predicative locking and replacing it with preemptive locking. The idea behind preemptive locking is simple: higher priority agents are *always* given the lock over lower priority agents regardless of the state of a mediation session. This change has serious implications on the need to exchange information that specifically supports predictive locking. One of these changes is that, with the exception of neighbors, links no longer need to be bi-directional. Another implication is that non-neighboring agents only need to exchange their current value when a mediation session is about to start. As the results in Section V show, these changes reduce messaging and improve the privacy of the agents. This technique is similar to the busy-waiting method proposed by Grinshpoun and Meisels [7], however differs because the agents perform useful work during the waiting period.

Another design change to the protocol is the removal of the **wait!** message. Instead, agents always return **evaluate!** messages in response to a request with a special flag indicating whether or not the agent has gotten the lock. Unlike the previous version of APO, which conducted session with partial information, or the version proposed by Grinshpoun and Meisels [7], which canceled incomplete sessions, this technique improves parallel performance because it allows the mediators to make decision based on more accurate information.

This work also investigates the impact of replacing the original solver in APO to address efficiency issues that have been previously reported for problem with large domains [4], [7]. One possible explanation for these findings is that the branch-and-bound solver used by the protocol performs an abnormally high number of constraint checks as the domain of the variables increase [11]. One suitable candidate to replace the solver is the Forward Checking, Conflict-directed Backjumping (FC-CBJ) [8] satisfiability solver. Although currently not the "best" algorithm, FC-CBJ is considerable more efficient that a simple backtracking algorithm because it combines prospective (forward checking) and retrospective (conflict direct backjumping) techniques to rapidly abandon and prune branches in the search tree.

Because FC-CBJ is a satisfiability solver, it suffers from the same drawbacks found in previous attempts as discussed in Section III-B. To limit the impact of this change, several variable and domain order heuristics [12] were introduced. The first heuristic statically arranges the domain elements of the variables based on the amount of external conflict they would create if selected. By arranging them in ascending order (from least to most), the first values to be tried when labeling a variable will result in the least external conflict. However, the impact of this heuristic is influenced by the order that the variables are labeled because variables that are assigned first, constrain the values of variables lower in the search tree. We mitigated this problem by initially ordering the variables based on the average amount of external conflict they create. Because only variables on the edge of the centralize subproblem have external constraints, they end up at the top of the search tree and are less likely to be changed. Overall this promotes cohesion. However, unlike the domain elements, the variable order does not remain static. We also utilize a dynamic min-domain heuristic during the search to both ensure the comparison between APO and AFC-CBJ is fair and to improve the efficiency of the problem solving process.

```
procedure choose_solution
  search for a solution s using FC-CBJ;
  if ¬∃s that satisfies the constraints do
    broadcast no solution;
  if mediator = x_i do
    mediator ← null;
    d_i ← d'_i ∈ s;
  end if;
  generate < pairs >;
  for each x_j ∈ preferences do
    if lock(x_j) = true do
      update agent_view for x_j
    if d'_j ∈ s violates an x_k ∧ x_k ∉ good_list do
      send_init_req (x_k);
    send (accept! d'_j, x_i, p_i, m_i, < pairs >) to x_j;
  end do;
  check_agent_view;
end;
```

Figure 6. APO-FCCBJ procedure for choosing a solution during an APO mediation.

### A. Soundness and Completeness

The soundness and completeness proofs follow the basic format of the proofs given in both Mailler and Lesser [1] and Grishpoun and Meisels [7]. As such, many of the components of the proof refer to those publications. The important changes are highlighted in this paper.

*Theorem 1:* **Neighboring** agents have bi-directional links.

*Proof:* In the initialization method, agents send linking requests to all of their neighbors. If a linking request is sent, it will be received, and a response will be generated. The response establishes the link. ■

*Theorem 2:* The new APO cannot deadlock.

*Proof:* A deadlock is a state where an agent has a conflict and wants to mediate, but instead does nothing. Assume that an agent $x_i$ enters a deadlock. This means that agent $x_i$ wants to mediate, but does not because it is in a stable state. One possibility is that $x_i$ has initiated a mediation session by sending **evaluate?** messages to the agents in its $good\_list$. Assuming communications are reliable, after a finite amount of time, $x_i$ will receive an **evaluate!** message from all of the agents and will initiate a mediation session. This contradicts the assumption that $x_i$ was in a stable state.

Another possibility is that $x_i$ wants to mediate, but does not send the requests. This can only occur if $x_i$ wants to mediate because it has a conflict and is expecting one of its neighbors to mediate instead. This is actually a simplified case of the proof presented in Mailler and Lesser [1]. ■

*Theorem 3:* The new APO algorithm is sound. i.e. it reaches a stable state only if it has found a solution or no solution exists.

*Proof:* Identical to the proof presented in Grinshpoun and Meisels [7] ■

*Theorem 4:* If there exists agents that holds all of the variables in their $good\_list$ and desire to mediate, then one of these agents will perform a mediation session.

*Proof:* There are two cases to consider.

**Case 1:** One agent has centralized the entire problem and wants to mediate. Consider an agent $x_i$ that holds all of the variables in its good_list and wants to mediate. Since $x_i$ is the only agent to have centralized all of the variables, it also has the highest priority and will start a mediation session by sending out **evaluate?** messages to all of the agents. Upon receiving these messages, all of the agents will preempt any session they may be in and give the lock to $x_i$. Since their are no higher priority agents, $x_i$ cannot be preempted and will perform a successful mediation session.

**Case 2:** More than one agent holds the entire graph and wants to mediate. In this case, ties are broken using the agents' index. Consider the one with the highest index among these agents and apply the same proof as case 1. ■

*Theorem 5:* If an agent holding all of the variables in their $good\_list$ performs a mediation session than the algorithm reaches a stable state.

*Proof:* Identical to the proof presented in Grinshpoun and Meisels [7] ■

*Theorem 6:* Agents cannot perform infinite value changes without performing a mediation session.

*Proof:* Identical to the proof presented in Grinshpoun and Meisels [7] ■

*Theorem 7:* If no agent will mediate or desire to mediate the algorithm reaches a stable state.

*Proof:* Nearly identical to the proof presented in Grinshpoun and Meisels [7] with the exception that agents do not send **wait!** or **cancel!** messages. They do, however send two types of **init** messages with a request triggering a single reply and the reply resulting in a call to the **check_agent_view** procedure. The rest of the proof remains the same. ■

*Theorem 8:* After $n$ mediation session, at least one of the good_lists grows, or the algorithm reaches a stable state.

*Proof:* Identical to the proof presented in Grinshpoun and Meisels [7] ■

*Theorem 9:* If there is a group of agents that desire to mediate, eventually a mediation session will be successful.

*Proof:* Assume that there is a group of agents that wants to mediate, but none of them are able to conduct a session with all of the agents they desire to mediate with. This means that every one of the mediators had at least one agent that refuses to give them a lock. However, since the mediators have an absolute order, this means that one of them is the highest priority mediator of the group. Because it is highest priority, when its **evaluate?** messages are received by the agents it wishes to mediate with, it must have been higher priority than any other mediator with the lock. Since the mediator is the highest priority, it must have received all of the locks and therefore will conduct a successful session, contradicting the assumption. ■

*Theorem 10:* The new APO algorithm is complete. i.e. if

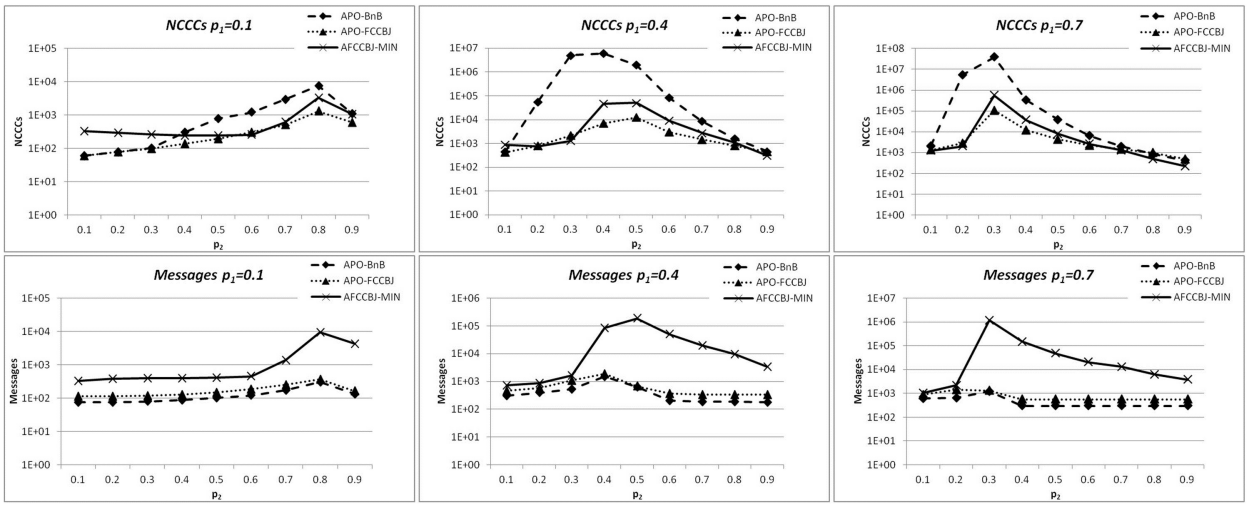


Figure 7. Results for random DCSP problems with $n = 20$, $k = 10$ of various density and tightness.

a solution exists the algorithm will find it otherwise it will report no solution.

*Proof:* Identical to the proof presented in Grinshpoun and Meisels [7] ■

## V. Empirical Evaluation

To evaluate the impact of the changes to APO, we conducted two separate sets of tests using a distributed CSP domain developed in the Farm multi-agent systems simulator [13]. In the first set of experiments we generated random DCSP problems with $n = 20$ variables, each with a domain size of $k = 10$, and varied both the density $p_1 = \{0.1, 0.4, 0.7\}$ and tightness $p_2 = \{0.1 - 0.9\}$ of the constraints. These particular values were chosen to replicate the experimental conditions of the evaluation conducted by Grinshpoun and Meisels [7].

For each combination of setting for $p_1$ and $p_2$, we created 100 instances. We then ran the original APO (denoted APO-BnB), the new APO (denoted APO-FCCBJ), and the AFCCBJ (denoted AFCCBJ-MIN) protocols. Each algorithm was given identical problems with the same initial variable assignments to minimize variance. The algorithms were allowed to run until completion. In total, 8,100 runs were conducted.

In the second set of experiments, we measured the scalability of all three protocols using randomly generated 3-coloring problems. We chose to replicate the experimental conditions of the original APO paper [1]. In that work, the authors conducted tests using 3 different edge densities that represent underconstrained ($d = 2.0$), phase transition ($d = 2.3$), and overconstrained ($d = 2.7$) problems. For each density setting, we varied the size of the problem from $n = 15$ to $n = 90$ variables in steps of 15 and generated 30 instances. In total, 1,620 tests were run. During all of the tests we measured the number of messages and NCCCs [5] used by the algorithms.

### A. Efficiency

The results of the experiments using random DCPSs can be seen in Figure 7, which are presented using a semi-log scale. It is easy to see in these graphs that both APO-BnB and APO-FCCBJ use over two orders of magnitude fewer messages than AFCCBJ-MIN in the phase transition regions. It is also quite clear that APO-BnB uses a huge number of NCCCs for difficult problem instances reaching nearly 39 million NCCCs on high density problems in the phase transition. However these graphs make it apparent that APO-FCCBJ is considerably more efficient that APO-BnB, using 2,285X (4,993,234 versus 2,185) fewer NCCCs at $p_1 = 0.4$, $p_2 = 0.3$ and 1,873X (38,608,845 versus 107,458) fewer NCCCs at $p_1 = 0.7$, $p_2 = 0.2$ . In addition, APO-FCCBJ uses the same or fewer NCCCs than AFCCBJ-MIN and in the phase regions is actually 2.5X (3,299 versus 1,304) more efficient at $p_1 = 0.1$, $p_2 = 0.8$, 6.6X (45,558 versus 6,902) more efficient at $p_1 = 0.4$, $p_2 = 0.4$, and 5.2X (560,270 versus 107,458) more efficient at $p_1 = 0.7$, $p_2 = 0.3$. Overall, APO-FCCBJ is 1.5X more efficient at $p_1 = 0.1$, 3.6X more efficient at $p_4 = 0.4$ and 4.7X more efficient at $p_1 = 0.7$. We should note that the results that we obtained for AFCCBJ-MIN closely match the values reported by Meisels and Zivan [6].

The 3-coloring results can be seen in Figure 8. The graphs indicate that AFCCBJ-MIN has an exponential growth rate in the number of messages as the problem size increases whereas both APO-BnB and APO-FCCBJ are nearly polynomial. Again, this was reported in the original APO results as well as in subsequent studies [1], [4], [7]. What can also be seen in these graphs is that all of the protocols use an exponential number of NCCCs. However, the growth rate for

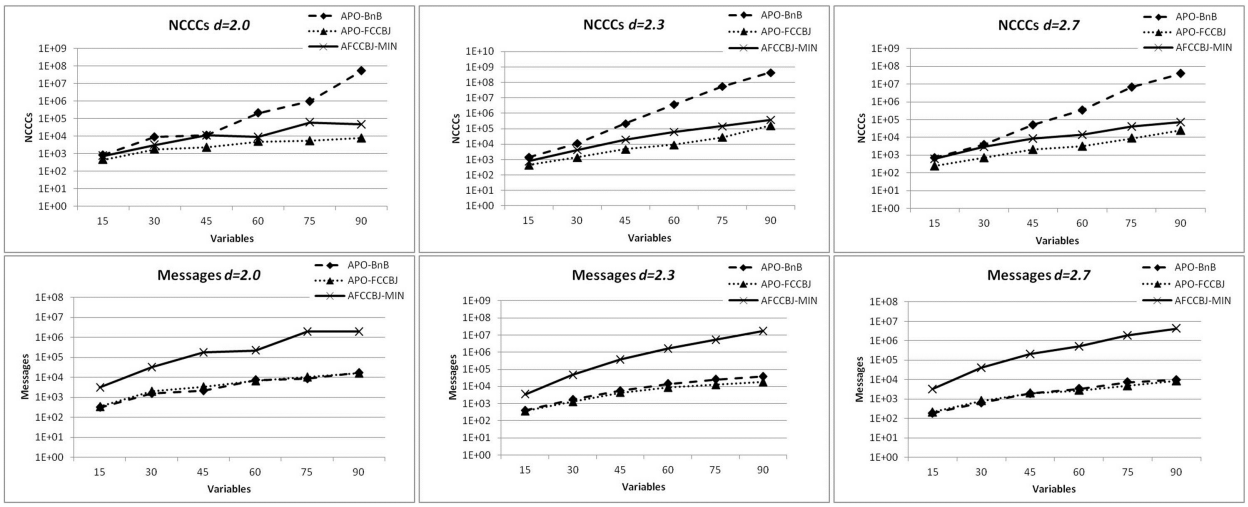


Figure 8. Scalability results using 3-coloring problems of various size and density.

AFCCBJ-MIN is higher than that of APO-FCCBJ. APO-FCCBJ, on average is 5.8X more efficient on graphs of density $d = 2.0$, 3.0X on $d = 2.3$, and 3.5X on $d = 2.7$.

There is one odd anomaly in the results for AFCCBJ-MIN at $d = 2.0$ and $n = 75$ that bares mentioning. We investigated the cause of this sudden increase in messages and NCCCs and found that one of the 30 low density instances was unsatisfiable. This one instance caused a dramatic increase in the search time of the protocol because, being low density, it could not easily prune branches of the search tree.

*Overall, APO-FCCBJ uses fewer communication and computational resources than APO-BnB and AFC-CBJ to solve random DCSP and distributed 3-coloring problems.*

### B. Privacy

The results of changing the centralized solver had very little impact on the size of the largest problem centralized in any one agent when comparing APO-FCCBJ to APO-BnB. For example, on low density ($p_1 = 0.1$) 20 variable random CSPs in the phase transition ($p_2 = 0.8$) APO-FCCBJ centralized 10 variables on average compared to 9 for APO-BnB. For medium and high density CSPs, APO-FCCBJ centralized on average one additional variable compared to APO-BnB. In the graph coloring domain, the change minimally increases the centralization from approximately 60% to 63% with the largest increase occurring on under-constrained problems, which went from 34% to 53%. The increase for underconstrained problems can be attributed to the large number of satisfying solutions found in low density subproblems. This makes it is less likely that the first satisfying solution creates the minimal amount of additional linking.

However, looking only at the largest centralization hardly tells the entire story. We conducted another set of experiments to determine if the changes to the protocol had, in any way, altered the distribution of messages or the linking pattern of the agents. Within the 3-coloring domain, we generated problems with $n = 60$ variables within the three regions of the phase transition. This time we counted the number of each message type exchanged during the runs and at the end of the run determined the additional linking done by each of the agents. The results of this investigation can be seen in Figure 9.

Across all three regions of the phase transition, we found a significant decrease in the amount of **ok?** messages being exchanged. This can be easily explained because this message type is no longer exchanged between non-neighboring agents. This finding shows that the privacy of agents has been improved because although they may have revealed their constraints, they no longer need to keep distant agents up to date on their current status.

The second set of graphs shows a histogram of the links added by the agents. As the graphs show, the distribution of linking had also changed. The histograms show that fewer agents overall have additional links. Combined with the previous results this shows that APO-FCCBJ centralizes in a much more controlled manner and that the centralization only occurs in fewer agents. So, although their is a slight increase in the number of variables centralized in a single agent, overall the constraints are being shared with fewer agents and less information is being exchanged once a link is established. These changes improve the algorithm's privacy [14].

## VI. Conclusion

In this paper we revisited the design and implementation of the APO protocol. Using a more detailed understanding of the requirements needed to ensure the protocol's soundness and completeness, we significantly modified to the control

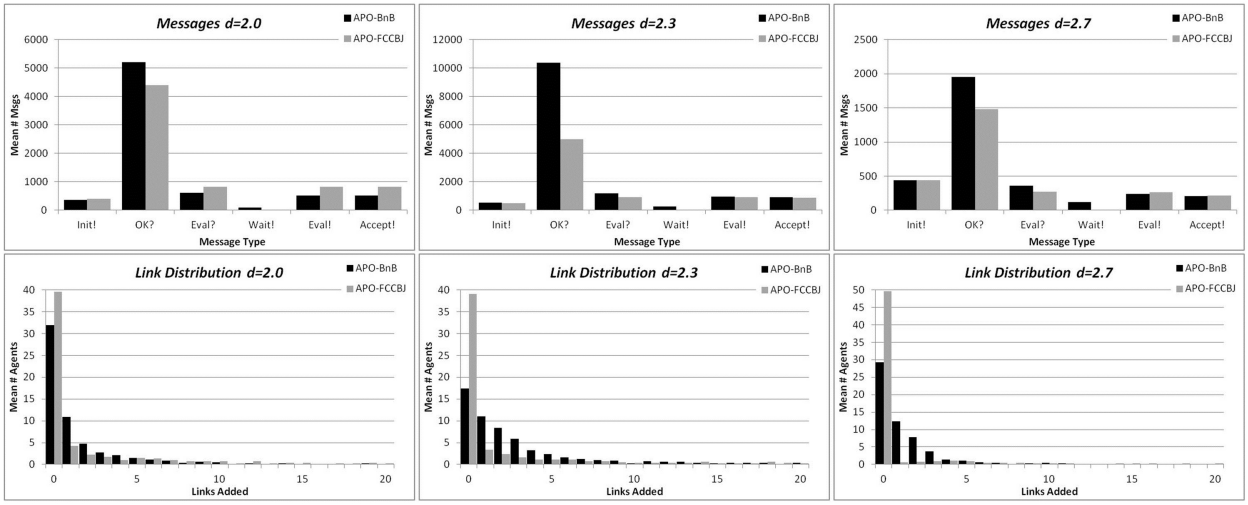


Figure 9. Comparison of message usage and linking of APO-BnB and APO-FCCBJ.

and information sharing techniques used by APO. In addition, we also replaced its central solver to improve its efficiency. We then showed that the new version of APO remains sound and complete and is vastly more efficient by comparing it to the AFC-CBJ protocol on random DCSP and 3-coloring problems. These results show that on nearly every case tested the APO protocol outperforms the AFC-CBJ protocol using every efficiency metric we measured. In addition, the resulting protocol provides better privacy to the agents because information and centralization are more tightly controlled.

## REFERENCES


[1] R. Mailler and V. Lesser, "Asynchronous Partial Overlay: A new algorithm for solving distributed constraint satisfaction problems," *Journal of Artificial Intelligence Research*, vol. 25, pp. 529–576, 2006.

[2] ——, "Solving distributed constraint optimization problems using cooperative mediation," in *Proceeding of AAMAS-2004*, 2004, pp. 438–445.

[3] M. Yokoo, "Asynchronous weak-commitment search for solving distributed constraint satisfaction problems." in *Proceedings of the First International Conference on Principles and Practice of Constraint Programming (CP-95)*, ser. Lecture Notes in Computer Science 976. Springer-Verlag, 1995, pp. 88–102.

[4] M. Benisch and N. Sadeh, "Examining DCSP coordination tradeoffs," in *Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multiagent Systems*, 2006.

[5] 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*, 2002, pp. 86–93.

[6] A. Meisels and R. Zivan, "Asynchronous forward-checking for DisCSPs," *Constraints*, vol. 12, pp. 131–150, 2007.

[7] T. Grinshpoun and A. Meisels, "Completeness and performance of the APO algorithm," *Journal of Artificial Intelligence Research*, vol. 33, pp. 223–258, 2008.

[8] P. Prosser, "Hybrid algorithms for the constraint satisfaction problem," *Computational Intelligence*, vol. 9, no. 3, pp. 269–299, 1993.

[9] M. Yokoo and E. H. Durfee, "Distributed constraint optimization as a formal model of partially adversarial cooperation," University of Michigan, Ann Arbor, MI 48109, Tech. Rep. CSE-TR-101-91, 1991.

[10] F. Bacchus and P. van Beek, "On the conversion between non-binary constraint satisfaction problems," *Proceedings of the 15th Nat'l/10th Conf. on Artificial Intelligence/Innovative Applications of Artificial Intelligence*, pp. 311–318, 1998.

[11] E. C. Freuder and R. J. Wallace, "Partial constraint satisfaction," *Artificial Intelligence*, vol. 58, no. 1–3, pp. 21–70, 1992.

[12] R. M. Haralick and G. L. Elliott, "Increasing tree search efficiency for constraint satisfaction problems," *Artificial Intelligence*, vol. 14, pp. 263–313, 1980.

[13] B. Horling, R. Mailler, and V. Lesser, "Farm: A scalable environment for multi-agent development and evaluation," in *Advances in Software Engineering for Multi-Agent Systems*, 2004, pp. 220–237.

[14] R. T. Maheswaran, J. P. Pearce, E. Bowring, P. Varakantham, and M. Tambe, "Privacy loss in distributed constraint reasoning: A quantitative framework for analysis and its applications," *Journal of Autonomous Agents and Multi-Agent Systems*, vol. 13, pp. 27–60, 2006.