

# **Examining DCSP Coordination Tradeoffs**

**Michael Benisch and Norman Sadeh**

December 2005  
CMU-ISRI-05-140

Institute for Software Research International  
School of Computer Science  
Carnegie Mellon University  
Pittsburgh, PA 15213

The research that lead to the development of the software described in this document has been funded by the National Science Foundation under ITR Grant 0205435 and under IGERT grant 9972762.

**Keywords:** Distributed Constraint Satisfaction, Multi-Agent Coordination, Multi-Agent Systems

## Abstract

Distributed Constraint Satisfaction Problems (DCSPs) provide a model to capture a broad range of cooperative multi-agent problem solving settings. Researchers have generally proposed two different sets of approaches for solving DCSPs, backtracking based approaches, such as Asynchronous Backtracking (ABT), and mediation based approaches, such as Asynchronous Partial Overlay (APO). These sets of approaches differ in the levels of coordination employed during conflict resolution. While the computational and communication complexity of the backtracking based approaches is well understood, the tradeoffs in complexity involved in moving toward mediation based approaches are not. In this paper we comprehensively reexamine the space of mediation based approaches for DCSP and fill gaps in existing frameworks with new strategies. We present different mediation session selection rules, including a rule that favors smaller mediation sessions, and different mediation strategies, including a decentralized hybrid strategy based on ABT. We present empirical results on solvable 3-coloring and random binary DCSP problems, that accurately capture the computational and communication tradeoffs between ABT and various mediation based approaches. Our results confirm that under some circumstances the newly presented strategies dominate previously proposed techniques.



# 1 Introduction

Distributed Constraint Satisfaction Problems (DCSPs) provide a formalism for representing problems where different agents are each responsible for instantiating different sets of variables subject to constraints. The agents aim to assign values to their variables, such that all constraints are satisfied. Many real-world scenarios can be modeled as DCSPs, such as supply chain coordination [10], product co-design, and distributed meeting scheduling [8].

There have been several distributed algorithms developed for solving DCSPs, each with particular strengths and weaknesses. The DCSP algorithm that has received the most attention is an asynchronous version of Constraint Backtracking (CBT), called Asynchronous Backtracking (ABT) [12]. The original proposal for ABT has seen several improvements [3, 2, 14], and is considered one of the most effective techniques for solving DCSPs.

Recently the multi-agent systems community has become interested in DCSP algorithms for modeling agent coordination tasks. This idea was introduced by Mailler and Lesser in the context of an algorithm called Asynchronous Partial Overlay (APO) [5]. In APO agents involved in a conflict select a mediator to solve a centralized version of a sub-problem capturing key elements of that conflict.

APO has been shown to involve significantly less communication than backtracking based techniques, such as ABT, by reducing the amount of unsuccessful instantiation attempts that repeatedly violate the same constraints (*thrashing*) [5]. On the other hand, in contrast with ABT, current versions of mediation based techniques have failed to fully exploit opportunities for concurrent processing. This tradeoff has not been explicitly identified in prior experimental investigations due to somewhat coarse computational complexity measurements.

In this paper we measure computational complexity based on a finer metric recently proposed by Meisels *et. al.* [6], that involves counting non-concurrent constraint checks. Comprehensive testing based on this finer metric reveals tradeoffs that had not been captured previously. Our results involve solvable 3-coloring problems, as well as random binary DCSPs. These results have motivated us to develop a hybrid technique that reconciles the tradeoffs we have identified. The basic intuition of our hybrid technique is to avoid thrashing on solvable problems through coordinated mediation, while maximizing concurrent work in the mediation process itself.

In Section 2 we introduce the DCSP formally, and discuss previous work on measuring problem solving complexity of DCSP algorithms. In Section 3 we discuss ABT and APO, along with new variations including our decentralized hybrid. Experimental results are presented and discussed in Section 5, and concluding remarks are provided in Section 6.

## 2 Background

### 2.1 DCSP Definition

The distributed constraint satisfaction problem was first discussed by Sycara *et. al.* and Yokoo *et.al.* as a way of formalizing Cooperative Distributed Problem Solving [12, 10]. A DCSP is formally defined as a constraint satisfaction problem (CSP) of the following form:

- a set of  $n$  variables,  $V = \{x_1, \dots, x_n\}$
- a set of discrete finite domains for each variable,  $D = \{D_1, \dots, D_n\}$
- a set of constraints  $R = \{R_1, \dots, R_m\}$  where each  $R_i(d_{i1}, \dots, d_{ij})$  is a predicate on the Cartesian product of the domains of all the variables referenced by that constraint. The constraint is said to be *satisfied* if the assignments of each referenced variable satisfy the constraint.

An agent,  $i$ , is said to *know* about a particular set of variables,  $V_i$ , and constraints,  $C_i$ . We will call the group of agents that agent  $i$  is connected to by constraints  $i$ 's neighborhood,  $N_i$ . The goal of the agents is to instantiate their variables so that all constraints are satisfied. As in other work, for the sake of simplicity, we restrict our presentation to situations where each agent controls (“owns”) a single variable and knows about all the constraints that refer to that variable. We further limit our discussion to binary constraints (constraints between two variables). It has been shown that more general problems can be reduced to problems that conform to these restrictions. In Section 6 we briefly discuss why the trends observed in this study should extend to these more complex classes of DCSPs.

### 2.2 Related Work

The original descriptions of ABT and APO appear in [12] and [5] respectively. In empirical studies of APO and backtracking based algorithms, such as ABT and Asynchronous Weak Commitment [11] (AWC), computational complexity measurements were often based on somewhat coarse performance metrics. In particular, these experiments typically did not differentiate between steps that required significantly different amounts of computation, e.g. equally counting as a single step a consistency check in AWC and a full backtracking search in a mediation session. Because they also focused primarily on solvable 3-coloring problems these experiments also raised the question of whether they were representative of other problem classes. In this paper we revisit the coordination tradeoffs involved in solving DCSPs using finer computational metrics and a more extensive class of problem instances - both solvable 3-coloring problems and random binary DCSPs. Our experiments reveal a richer set of performance tradeoffs than had been reported earlier. In addition, we present novel search configurations that aim to reconcile these tradeoffs.

Our empirical analysis uses a metric introduced by Meisels *et.al.* [6] that records the number of non-concurrent constraint checks (NCCCs) associated with different distributed search procedures. This metric distinguishes between complex computational steps involving multiple constraint checks, and much simpler ones. Constraint checks are the preferred computational unit used to measure performance of centralized constraint processing algorithms (see for example [9]). By counting non-concurrent checks the NCCCs metric is akin to the notion of a make-span or throughput measurement in scheduling [4].

The results presented in this paper build on our preliminary work comparing different mediation session selection rules for APO [1].

### 3 Basic Backtracking and Mediation Algorithms

This section outlines key features of ABT and APO, focusing on elements that impact computation and communication complexity.

#### 3.1 Asynchronous Backtracking

ABT is an asynchronous implementation of the traditional backtracking search approach to solving centralized CSPs.

A summary of the procedures involved in the original description of ABT is shown in Figure 1. The algorithm begins with an initialization process, during which each agent  $i$  is given a static priority,  $p_i$  (typically based on lexicographic order). Agent  $i$  then arbitrarily assigns a value,  $d_i$ , to its variable,  $x_i$ , and broadcasts it to all lower priority neighbors. When agent  $i$  receives an assignment message from another agent,  $j$ , it stores the assignment as a tuple in the set called its *agentview*. Agent  $i$  then checks the consistency of its *agentview* and its own assignment. When it finds that its own assignment violates a constraint it tries to assign a value to its variable that resolves consistency. If it cannot find such an assignment it begins the backtrack process, and informs a higher priority agent that its *agentview* is *nogood* (leads to an unsatisfiable situation for the agent)<sup>1</sup>. The algorithm proceeds until all of the agents have variable assignments consistent with their *agentviews*, or one of the agents discovers that the problem is infeasible. For a full description of the ABT algorithm along with proofs of completeness and correctness, the reader is directed to its original description in [12].

An agent executing ABT uses constraint checks<sup>2</sup> to determine the consistency of its variable assignment. Because the agent knows only about the assignments of higher priority agents (as defined in its *agentview*), this amounts to checking each constraint with a higher

---

<sup>1</sup>We are aware that an agent can potentially resolve the subset of its *agentview* that resulted in the conflict. However, this has been shown to provide little benefit.

<sup>2</sup>Note that this refers to constraint checks in general, whether or not these checks are concurrent.

---

```

procedure init()
  calculate priority,  $p_i$  (usually lexicographic order);
  choose  $d_i \in D_i$  arbitrarily; set  $x_i \leftarrow d_i$ ;
   $\text{agent\_view}_i \leftarrow \emptyset$ ;
  broadcast();

when received ok?( $j, d_j$ )
  update_agent_view( $j, d_j$ );
  check_agent_view();

procedure update_agent_view( $j, d_j$ )
  if  $\text{agent\_view}_i$  contains assignment for  $j$  then
    remove assignment of  $j$  from  $\text{agent\_view}_i$ ;
  end if
   $\text{agent\_view}_i \leftarrow \text{agent\_view}_i \cup \{ \langle j, d_j \rangle \}$ ;

procedure check_agent_view()
  if  $\neg \text{consistent}(\text{agent\_view}_i \cup \{ \langle i, x_i \rangle \}, C_i)$  then
    if choose_value() then
      broadcast();
    else
      resolve_conflict();
    end if
  end if

procedure broadcast()
  for agent  $j \in N_i \mid p_j < p_i$  do
    send ok?( $i, x_i$ ) to agent  $j$ ;
  end for

procedure choose_value()
  for  $d_i \in D_i$  do
    if consistent( $\text{agent\_view}_i \cup \{ \langle i, d_i \rangle \}, C_i$ ) then
       $x_i \leftarrow d_i$ ; return  $\top$ ;
    end if
  end for
  return  $\perp$ ;

procedure resolve_conflict()
  if  $\text{agent\_view}_i \neq \emptyset$  then
    find agent  $j \in \text{agent\_view}_i$  with lowest priority,  $p_j$ ;
    send nogood( $i, \text{agent\_view}_i$ ) to agent  $j$ ;
  else
    No solution!
  end if

when received nogood( $j, \text{agent\_view}_j$ )
  for agent  $k$  in  $\text{agent\_view}_j$  and not in  $\text{agent\_view}_i$  do
    let  $d_k$  be the value of agent  $k$  in  $\text{agent\_view}_j$ ;
    update_agent_view( $k, d_k$ );
    request that  $k$  add  $i$  to  $N_k$ ;
  end for
  let  $c_j$  be the constraint that  $\text{agent\_view}_j$  is nogood;
   $C_i \leftarrow C_i \cup \{c_j\}$ ;
  check_agent_view();

```

---

Figure 1: Procedures Defining the Behavior of an ABT Agent  $i$



---

```

when received ok?( $j, d_j$ )
  update_agent_view( $j, d_j$ );
  prune_stale_nogoods();
  check_agent_view();

procedure prune_stale_nogoods()
  for nogood constraint  $c^{ng} \in C_i$  do
    let agent_view $_c$  be the agentview that created  $c^{ng}$ ;
    if agent_view $_c \not\subseteq$  agent_view $_i \cup \{\langle i, x_i \rangle\}$  then
       $C_i \leftarrow C_i \setminus \{c^{ng}\}$ ;
    end if
  end for

when received nogood( $j, \text{agent\_view}_j$ )
  for agent  $k$  in agent_view $_j$  and not in agent_view $_i$ 
  do
    let  $d_k$  be the value of agent  $k$  in agent_view $_j$ ;
    update_agent_view( $k, d_k$ );
    request that  $k$  add  $i$  to  $N_k$ ;
  end for
  if agent_view $_j \subseteq$  agent_view $_i \cup \{\langle i, x_i \rangle\}$  then
    let  $c_j$  be the constraint that agent_view $_j$  is no-
    good;
     $C_i \leftarrow C_i \cup \{c_j\}$ ;
    check_agent_view();
  end if

```

---

Figure 2: Revisions to ABT for Nogood Pruning

priority agent. Notice that the original description of ABT involves recording each received nogood as a constraint (see “**when received nogood**” in Figure 1). As a result, each learned nogood is among the set of constraints that must be checked for consistency. Since the storage of nogoods can grow exponentially on harder problem instances, checking nogoods as constraints can significantly increase the number of NCCCs used by ABT. It has been shown, however, that agents can avoid much of the constraint checking work associated with nogoods by eliminating them from storage once they become *stale* (they refer to a branch of the search tree that has already been discarded) [2]. The necessary revisions to ABT for pruning stale nogoods are given in Figure 3.1. When measuring the computational requirements of this revised version of ABT, it is important to take into account the additional checks that have been introduced to determine whether or not a nogood is stale. However, because the number of stale nogoods tends to grow exponentially, the savings typically far outweigh this additional computational cost. We would like to note additionally, that this improvement cannot be used in AWC because priorities are assigned dynamically and stale nogoods may become relevant again. This makes AWC less competitive when measuring computational requirements based on consistency checks. For this reason we do not report experiments with AWC in this paper.

Another configuration of ABT involves having agents read all queued messages before trying to re-assign their variables. In contrast to checking consistency after each incoming assignment update or nogood, agents processes all the information they have received before performing their next consistency check (“**procedure check\_agent\_view**”). This small variation has recently been reported to provide significant improvements [13].

---

```

procedure broadcast()
  for all agents  $j \in N_i$  do
    send  $\text{ok?}(i, x_i)$  to agent  $j$ ;
  end for

procedure resolve_conflict()
   $\text{accepted}_i \leftarrow \emptyset$ ;
  for agent  $j \in N_i$  do
    send  $\text{invitation}(i)$  to agent  $j$ ;
     $\text{rsvp}_i \leftarrow \text{rsvp}_i \cup \{j\}$ ;
  end for

when received  $I : \{\text{invitation}(j_1), \dots, \text{invitation}(j_n)\}$ 
  /*  $I$  is a set of  $n$  invitations received at the same time */
  if agent  $i$  is already involved in a mediation session then
    send  $\text{reject\_invitation}(i)$  to all inviting agents;
  else
    find  $\text{invitation}(j_k) \in I$  specified by selection rule,  $R_{\prec}$ ;
    send  $\text{accept\_invitation}(i)$  to agent  $j_k$ ;
    send  $\text{reject\_invitation}(i)$  to all other inviting agents;
  end if

when received  $\text{accept\_invitation}(j)$ 
   $\text{accepted}_i \leftarrow \text{accepted}_i \cup \{j\}$ ;
   $\text{rsvp}_i \leftarrow \text{rsvp}_i \setminus \{j\}$ ;
  if  $\text{is\_empty}(\text{rsvp}_i)$  then
    mediate( $\text{accepted}_i$ );
  end if

when received  $\text{reject\_invitation}(j)$ 
   $\text{rsvp}_i \leftarrow \text{rsvp}_i \setminus \{j\}$ ;
  if  $\text{is\_empty}(\text{rsvp}_i)$  then
    mediate( $\text{accepted}_i$ );
  end if

```

---

Figure 3: Mediation DCSP Algorithm Framework (procedures not redefined are given in Figure 1)

### 3.2 Asynchronous Partial Overlay

A summary of the procedures involved in a basic mediation framework for DCSPs is given in Figure 3. Procedures that are not redefined are assumed to be identical to the description in Figure 1. The initialization process of a mediation based algorithm proceeds the same way as in ABT. However, when agent  $i$  broadcasts its value assignment, it does not consider priority and sends the assignment to all its neighbors. When agent  $i$  receives an assignment message it proceeds in the same fashion as ABT to update its *agentview* and check the consistency of relevant constraints. If agent  $i$  detects a conflict and cannot resolve it locally by changing the value of its variable it initiates a conflict resolution process. In a mediation based algorithm this involves agent  $i$  inviting all of its neighbors to join a mediation session - the size of the mediation session is defined by agent  $i$ 's neighborhood. Once all of agent  $i$ 's neighbors have each either accepted or rejected the invitation, a mediation session begins. A mediation session only involves the agents that have accepted the invitation. An agent can only accept to participate in one mediation session at a time. Agents choose which

invitation to accept according to a mediation session selection rule, which we denote  $R_{\succ}$ . Additional details of the APO algorithm, a specific mediation based implementation, can be found in [5].

In the following Section we review different possible mediation session selection rules and mediation procedures, and discuss their constraint checking requirements. Independently of the mediation session selection rule and mediation procedure, mediation based agents use constraint checks to determine the consistency of their variable assignments.

## 4 Mapping Mediation Dimensions

The space of possible mediation strategies has been the subject of limited investigation. In this paper we focus on two mediation dimensions that we have found to have a significant impact on the performance of mediation based algorithms, namely mediation session selection rules and mediation procedures themselves.

### 4.1 Mediation Session Selection Rules

The mediation session selection rule described in the original APO algorithm is biased toward the selection of larger mediation sessions. Preliminary work reported in [1] presented a computational model along with initial results suggesting that, under certain circumstances, focusing first on smaller sessions can yield significant performance improvements. Intuitively, because mediation sessions may involve full backtrack search, the complexity of large mediation sessions will tend to dominate that of smaller sessions. Accordingly, focusing first on large sessions will lead to unnecessary computational efforts, when smaller sessions are sufficient. In this study we explore the performance impact of mediation session size by examining the following two mediation session selection rules.

- $R_{\succ}^{\text{APO}}$ : this mediation session selection rule was suggested in the original definition of APO. Agents evaluate invitations based on the size of the mediation session they would lead to, and pick the invitation corresponding to the largest one.
- $R_{\prec}^{\text{IAPO}}$ : this mediation session selection rule is the inverse of the original selection rule. It instructs agents to choose the smallest mediation session.

These two rules can be implemented by including the size of the inviting agent’s neighborhood in each invitation. For all practical purposes the computational complexity of evaluating mediation session invitations using these two rules can be ignored. The evaluation can be performed by each agent in time linear in the number of invitations, which does not exceed the size of the agent’s neighborhood. This computation is typically dominated by the computation performed as part of the mediation procedure itself.

---

<pre> <b>when received</b> <code>accept_invitation</code><sup>BB</sup>(<math>j, C_j, D_j, L_j</math>)   <code>accepted</code><sub><math>i</math></sub> <math>\leftarrow</math> <code>accepted</code><sub><math>i</math></sub> <math>\cup</math> <math>\{ \langle j, C_j, D_j, L_j \rangle \}</math>;   ... </pre>	<pre> <b>when received</b> <code>accept_invitation</code><sup>ABT</sup>(<math>j</math>)   <code>accepted</code><sub><math>i</math></sub> <math>\leftarrow</math> <code>accepted</code><sub><math>i</math></sub> <math>\cup</math> <math>\{ j \}</math>;   ... </pre>
<pre> <b>when received</b> <code>accept_invitation</code><sup>BT</sup>(<math>j, C_j, D_j</math>)   <code>accepted</code><sub><math>i</math></sub> <math>\leftarrow</math> <code>accepted</code><sub><math>i</math></sub> <math>\cup</math> <math>\{ \langle j, C_j, D_j \rangle \}</math>;   ... </pre>	

---

Figure 4: Information Collected by Mediation Strategies

## 4.2 Mediation Procedures

The mediation procedure originally proposed with APO involved a *mediator* (the agent that sent the invitations) centralizing the sub-problem defined by the agents in the session, and solving it locally. The mediator used a branch and bound search to find a solution that minimized conflicts with agents outside of the session. In this Section we describe the original APO branch and bound mediation procedure, and propose two new mediation procedures (summarized in Figure 5). We introduce a backtrack search mediation procedure to explore the performance tradeoffs involved in finding a conflict minimizing solution. We also introduce a novel hybrid procedure where mediation is performed in a distributed fashion using ABT. This hybrid algorithm helps us explore the performance tradeoffs between concurrency and coordination. The different mediation procedures we examine require different information to accompany `accept_invitation` messages. The requirements are summarized in Figure 4 and described in detail along with the procedures below.

**Branch and Bound (APO-BB):** The branch and bound mediation strategy was the method originally proposed with APO. When using this mediation strategy, a mediator must gather all of the information about the local sub-problems of agents involved in the session (the agents that accepted the invitation) including:

- the set of constraints,  $C_j$ , that apply to agent  $j$ 's variable,
- the entire domain of agent  $j$ 's variable,  $D_j$ ,
- and a label function defined on agent  $j$ 's domain that indicates all of the agents known to be in conflict with each of  $j$ 's values,
 
$$L_j : d_j \rightarrow \{k_1, \dots, k_n \mid \text{agent } k_i \text{ will be conflicted with } d_j\}$$

This information is used by the mediator to perform a branch and bound search. This search finds a feasible solution, if one exists, to the sub-problem pertaining to agents that

---

```

procedure mediateBB(acceptedi)
  for  $\langle j, C_j, D_j, L_j \rangle \in \text{accepted}_i$  do
     $C \leftarrow C \cup \{C_j\}; D \leftarrow D \cup \{D_j\}; L \leftarrow L \cup \{L_j\};$ 
  end for
   $\langle P^*, b^* \rangle \leftarrow \text{b\&b}(\langle \emptyset, 0 \rangle, \langle \emptyset, \infty \rangle, 1, C, D, L)$ 
  broadcast-solution( $P^*$ )

procedure mediateBT(acceptedi)
  for  $\langle j, C_j, D_j \rangle \in \text{accepted}_i$  do
     $C \leftarrow C \cup \{C_j\}; D \leftarrow D \cup \{D_j\};$ 
  end for
   $P^* \leftarrow \text{cbt}(\emptyset, 1, C, D);$ 
  broadcast-solution( $P^*$ )

procedure mediateABT(acceptedi)
  let  $I$  be the sub-DCSP defined by agents in acceptedi;
  init() ABT on  $I$  (as described in Section 3.1);

procedure broadcast-solution( $P$ )
  if  $P \neq \emptyset$  then
    for  $\langle j, d_j \rangle \in P$  do
      send accept_value( $d_j$ ) to agent  $j$ ;
    end for
  else
    No solution!
  end if

when received accept_value( $d$ )
   $x_i \leftarrow d;$ 
  check_agent_view();

```

```

procedure b&b( $\langle P, b \rangle, \langle P^*, b^* \rangle, i, C, D, L$ )
  if  $|P| = |D|$  then
    return  $\langle P^*, b^* \rangle$ 
  end if
  for  $d \in D_i$  do
     $\langle P_d, b_d \rangle \leftarrow \langle \emptyset, \infty \rangle$ 
     $P' \leftarrow P \cup \{\langle i, d \rangle\}; b' \leftarrow b + |L_i(d)|$ 
    if  $b' < b^* \wedge \text{consistent}(P', C_i)$  then
       $\langle P_d, b_d \rangle \leftarrow \text{b\&b}(\langle P', b' \rangle, \langle P^*, b^* \rangle, i + 1, C, D, L)$ 
    end if
    if  $b_d < b^*$  then
       $b^* \leftarrow b_d; P^* \leftarrow P_d;$ 
    end if
  end for
  return  $\langle P^*, b^* \rangle$ 

procedure cbt( $P, i, C, D$ )
  if  $|P| = |D|$  then
    return  $P$ 
  end if
  for  $d \in D_i$  do
     $P' \leftarrow P \cup \{\langle i, d \rangle\};$ 
    if consistent( $P', C_i$ ) then
       $P^* \leftarrow \text{cbt}(P', i + 1, C, D)$ 
      if  $P^* \neq \emptyset$  then
        return  $P^*$ 
      end if
    end if
  end for
  return  $\emptyset$ 

```

---

Figure 5: Summary of Mediation Procedures

are part of the mediation session. The solution minimizes the number of constraints violated for agents *outside* of the session, which can be determined using the label functions. In terms of measuring computation, the branch and bound procedure uses constraint checks as it proceeds to construct all solutions that cannot be ruled out by bounding. However, it does not use constraint checks to determine how many external violations a partial or full solution results in. The number of external violations is simply the size of the set returned by the label function.

**Backtracking (APO-BT):** This mediation procedure proceeds exactly as the BB variant, except that the mediator attempts to find any feasible solution. The solution may not minimize conflicts with out-of-session agents. Using this mediation procedure, agents need not send fully labeled domains to the mediator<sup>3</sup>. In terms of measuring computation, the backtracking mediation process uses constraint checks in the same fashion as the branch and bound process, except it short-circuits computation as soon as a feasible solution is found.

**Asynchronous Backtracking (APO-ABT):** We introduce this mediation procedure as a novel hybrid between APO and ABT where mediations are performed using the ABT protocol. This mediation strategy downplays the importance of the mediator, who no longer needs to collect local constraint information from the other agents involved. Because agents can be involved in one session at a time APO-ABT synchronizes the overall problem solving process, while exploiting opportunities for concurrency at the lower level. The ABT mediation process uses constraint checks as described in 3.1.

Any of these three mediation strategies can be used with either of the mediation session selection rules, making for a total of six proposed configurations. Also, each of the mediation procedures uses the values of agents at the time of the mediation as arbitrary starting values. This was suggested in the original specification of APO and helps to respect work done during earlier sessions.

## 5 Empirical Evaluation

### 5.1 Empirical Setup

The experiments in this paper are intended to compare the performance of the six mediation configurations outlined in the previous section (five of which represent novel configurations), and ABT. We present results from two different sets of experiments, one involving random solvable distributed 3-coloring (D3C) problems and another on random binary DCSP in-

---

<sup>3</sup>Agents may, however, need to inform the mediator of a conflict after assignments have been dictated. This allows the mediator to add an out-of-session agent to its neighborhood to ensure completeness.

stances. A D3C problem involves each agent choosing one of three colors. The problem is satisfied if no two agents connected by a constraint share the same color. A random binary DCSP problem involves arbitrary random constraints between pairs of agents. In all of our experiments all of the algorithms were provided with the same instances and initial random values for variables.

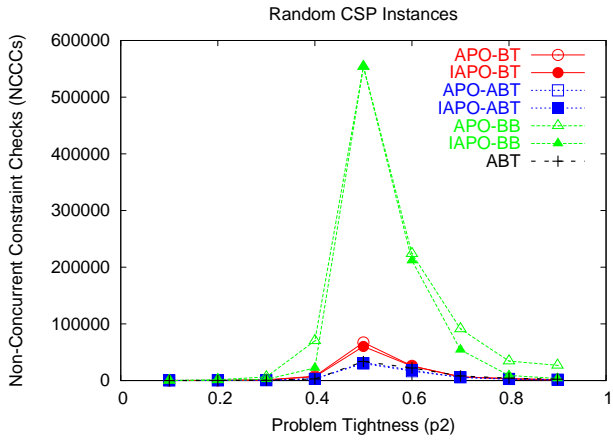
Random binary CSP instances are typically characterized by the number of variables,  $n$ , the number of values in each variable’s domain,  $k$ , the density of the constraint graph,  $p1$ , and the tightness of the constraints,  $p2$  [9]. Our random DCSP experiments use  $n = 10$ , and  $k = 10$ .  $p1$  specifies the probability that a constraint exists between any two agents. In this paper we fix  $p1 = 0.7$ , a relatively high density value, behaviors on problems with different values of  $p1$  result in similar observations. We vary  $p2 = 0.1$  to  $p2 = 0.9$ , which specifies the probability that two values in constrained agent’s domains are in conflict. We draw at least 100 instances for each value of  $p2$ , and about 500 instances near the phase transition where  $p2 \in [0.4, 0.6]$ . The “phase transition” refers to an area of the problem parameters where problems transition from being under-constrained, and thus easy to solve, to over-constrained but still difficult to prove infeasible [9].

The D3C instances are characterized by two parameters, the number of agents involved,  $n$ , and the number of constraints per agent  $m$ . Since we are looking at solvable D3C problems with no phase transition effects, we do not see a need to vary problem density and instead fix  $m = 2.7$  (typically considered high density). We vary the number of agents from  $n = 15$  to  $n = 60$  in increments of 9 (these values must be multiples of 3 to properly guarantee solvable instances). With these instances we examine how the algorithms scale to larger problems that are guaranteed to be solvable. We draw at least 100 instances for each value of  $n$  (although some algorithms are removed as they become incapable of scaling based on a 5 minute cpu-time cutoff per instance) according to the method provided in [7], for some larger values of  $n$  we draw up to 500 instances to ensure the statistical significance of our results.

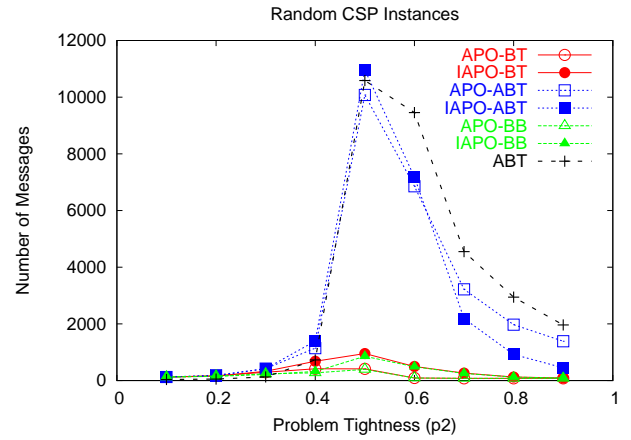
The results presented indicate the mean number of messages and non-concurrent constraint checks (NCCCs) used by each algorithm across the different simulations. The NCCCs are measured using the techniques described in Sections 3 and 4, and [6]. We provide 95% confidence intervals when necessary, results missing intervals can be assumed to be significant.

## 5.2 Results on Random CSP Instances

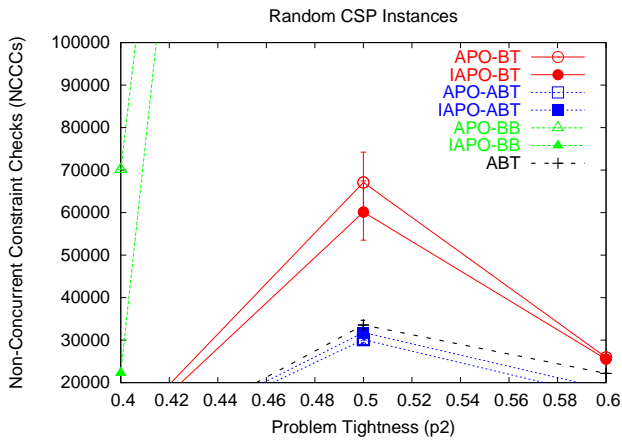
The NCCC results on random CSP instances are shown in Figures 6(a) and 6(c). Examining these results, we can see that the phase transition is clearly accentuated when  $p2 = 0.5$  for all the algorithms. Before this point problems are relatively easy to solve and there is little difference observed. When  $p2 = 0.5$  we can see that ABT, with the improvements described in Section 3.1, uses about 30,000 NCCCs on average, which is consistent with results reported



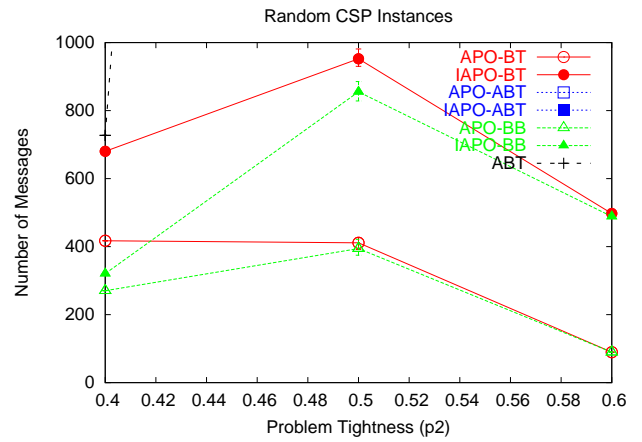
(a) Mean NCCCs



(b) Mean Messages



(c) Mean NCCCs with 95% confidence intervals (zoomed from Figure 6(a),  $p_2 = 0.4$  to  $p_2 = 0.6$ )



(d) Mean Messages with 95% confidence intervals (zoomed from Figure 6(b),  $p_2 = 0.4$  to  $p_2 = 0.6$ )

Figure 6: Performance on Random DCSP Instances, ( $n = 10, k = 10, p_1 = 0.7$ )



in [13]. In addition the APO-ABT and IAPO-ABT hybrids both use about the same amount of NCCCs as ABT itself (see Figure 6(c)). This is not surprising, since about half of the problems in this area require a mediation of the entire network to prove infeasibility, and the hybrid algorithms reduce to running ABT on the whole problem at that point. The branch and bound based algorithms, APO-BB and IAPO-BB, perform poorly during the phase transition, requiring about 200 times more NCCCs than ABT. This is due to the fact that many of these problems must be fully centralized with these procedures to prove infeasibility, and finding the solution that minimizes external conflicts amounts to exploring nearly the entire search tree during each mediation. The backtracking based mediation algorithms, APO-BT and IAPO-BT, require nearly 100 times fewer NCCCs than the branch and bound algorithms, showing that finding external conflict minimizing solutions costs a significant number of NCCCs and provides far less benefit on random problems. However, the ABT based algorithms use significantly fewer NCCCs than the backtracking ones, showing the cost of centralizing the problem solving effort.

The performance in terms of communication requirements on random instances is shown in Figures 6(b) and 6(d). These results are basically a mirror image of the NCCC results, with ABT and the ABT based mediation algorithms requiring on average about 30 times more messages than the other algorithms (again this is consistent with results reported for ABT in [13] and [5]). Additionally we can see from the results in Figure 6(d) that the mediation techniques with the IAPO preference rule use about twice as many messages as those with the APO preference rule. This is not surprising considering that the IAPO preference rule chooses smaller sessions and therefore needs more of them.

### 5.3 Results on Solvable D3C Instances

The NCCC results for solvable D3C instances are shown in Figure 7(a). These results provide some very interesting contrasts with the random CSP NCCC results. The algorithm that performs best on the solvable D3C instances by a large margin is our IAPO-BB algorithm. This algorithm was the only one capable of solving 60 agent problems within an allotted 5 cpu-minutes per instance, and it was able to solve smaller instances orders of magnitude faster than the other algorithms. Recall that on the random CSP instances the branch and bound based algorithms performed nearly 200 times *worse* than the other algorithms, yet on the high density solvable instances our IAPO-BB scales extremely well. What we are seeing is that on high-density solvable instances the benefit of finding solutions that minimize conflicts outside of a mediation session far outweighs the extra work necessary to do so.

Another interesting result on the solvable D3C instances is the benefit of the IAPO selection rule over the APO selection rule. All of the variants with our IAPO selection rule used significantly fewer NCCCs than their APO-based counterparts. Favoring smaller mediation sessions over larger ones, as IAPO does, helps mediators avoid solving unnecessarily large problems until absolutely needed. Our results suggest that on solvable D3C instances it is

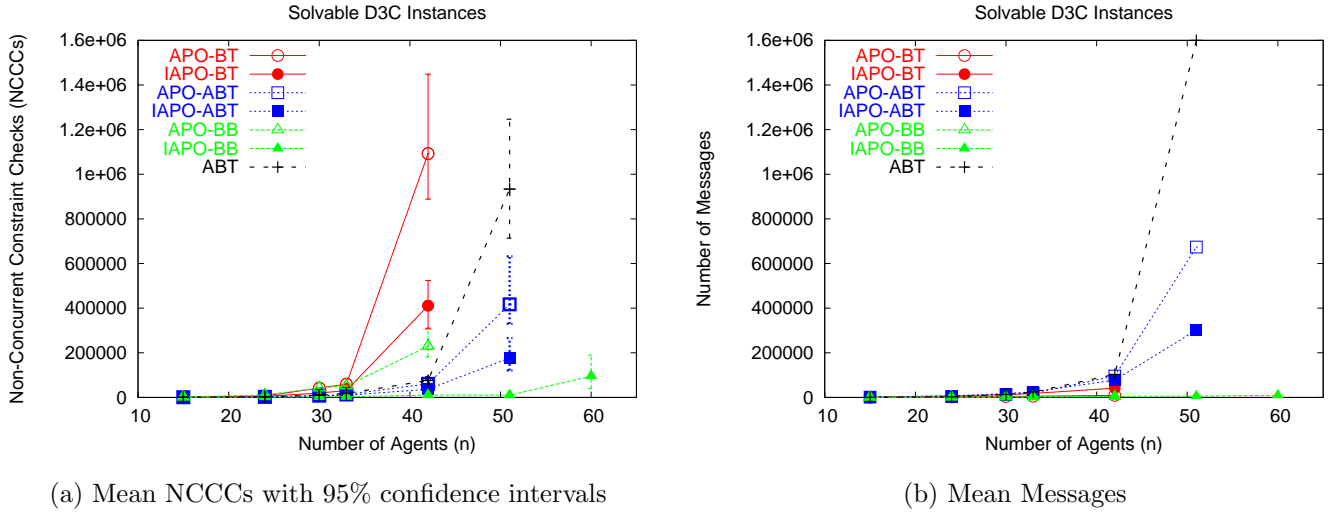


Figure 7: Performance on Solvable D3C Instances, ( $m = 2.7$ )

often not necessary to mediate over larger groups, and consequently IAPO avoids some of the unnecessary computation of APO. These findings are consistent with preliminary results we had reported in [1].

When we look at the handful of algorithms that were able to solve 51 agent instances we can see that only the ABT-based algorithms remain along with IAPO-BB. Among the ABT-based algorithms it is interesting to note that both our APO-ABT and IAPO-ABT hybrid algorithms require much fewer NCCCs than ABT itself on average (the IAPO-ABT algorithm requiring significantly fewer). This begins to show the benefits of partially synchronizing problem solving efforts on solvable D3C instances to cut back on the amount of backtracking in ABT. The fact that all three of these algorithms perform significantly better than the BT-based algorithms shows the benefit of distributing work load on larger problem instances.

When we look at the performance in terms of communication requirements for solvable D3C instances in Figure 7(b), it is not surprising to find that all of the centralized approaches use significantly fewer messages than the decentralized ones. What is interesting is the fact that our APO-ABT and IAPO-ABT hybrid algorithms both used fewer messages than ABT itself (recall these two used fewer NCCCs as well). Not only that, but the IAPO variant, IAPO-ABT, used fewer messages than the APO variant, whereas on the random instances the opposite was true. Both of these observations are due to the fact that the mediation based techniques required less backtracking than ABT and among the mediation based techniques, the IAPO variant required less backtracking than the APO variant. On the solvable D3C instances the decrease in backtracking during problem solving led to less communication as well.

## 6 Conclusion

In this paper we provided a comprehensive abstraction for the space of DCSP mediation based approaches. In the process we identified gaps in existing frameworks, and proposed new effective strategies to fill them. The new strategies include a mediation session selection rule that favors smaller mediation sessions, a centralized mediation approach that short-cuts a significant amount of excess computation on random instances, and a decentralized asynchronous mediation strategy based on ABT.

Our empirical results represent the first truly comprehensive view of the tradeoffs between backtracking and mediation based approaches for DCSPs. We tested the different algorithms on solvable 3-coloring and random binary DCSPs. Our results confirmed that our newly proposed strategies in this paper dominate previous techniques under some circumstances. Specifically, our mediator selection rule that favors smaller sessions, IAPO, dominates the previously used rule that favors larger ones, APO, in computational complexity across all instances. IAPO also dominates APO in communication complexity on solvable D3C instances. Additionally, our hybrid decentralized mediation strategy based on ABT is as good as ABT in communication and computation on random instances, and strictly better in both areas on solvable D3C instances. This suggests that the benefits of synchronization for DCSPs reported elsewhere [5, 13] may exist primarily on hard solvable instances.

## 7 Acknowledgements

The research reported in this paper has been funded by the National Science Foundation under ITR Grant 0205435. The authors would also like to thank Roie Zivan and Amnon Meisels for their help calibrating the NCCC metric.

## References

- [1] M. Benisch and N. Sadeh. How (not) to choose mediators for distributed constraint satisfaction. In *Proceedings of LSMAS at AAMAS'05*, 2005.
- [2] C. Bessiere, A. Maestre, and P. Messeguer. Distributed dynamic backtracking. In *Proceedings of DCR Workshop at IJCAI'01*, 2001.
- [3] Y. Hamadi. Distributed interleaved parallel and cooperative search in constraint satisfaction networks.
- [4] L. Lamport. The parallel execution of do loops. *Commun. ACM*, 17(2):83–93, 1974.

- [5] R. Mailler and V. Lesser. Using Cooperative Mediation to Solve Distributed Constraint Satisfaction Problems. In *Proceedings of Third International Joint Conference on Autonomous Agents and MultiAgent Systems (AAMAS 2004)*, volume 1, pages 446–453, New York, 2004. IEEE Computer Society.
- [6] A. Meisels, E. Kaplansky, I. Razgon, and R. Zivan. Comparing performance of distributed constraints processing algorithms. In *Proceedings of DCR Workshop at AAMAS'02*, 2002.
- [7] S. Minton, M. D. Johnston, A. B. Phillips, and P. Laird. Minimizing conflicts: A heuristic repair method for constraint satisfaction problems. *Artificial Intelligence*, 58(1-3):161–205, 1992.
- [8] P. J. Modi, M. Veloso, S. Smith, and J. Oh. Cmradar: A personal assistant agent for calendar management. In *Agent Oriented Information Systems, (AOIS) 2004*, 2004.
- [9] B. M. Smith and M. E. Dyer. Locating the phase transition in binary constraint satisfaction problems. *Artif. Intell.*, 81(1-2):155–181, 1996.
- [10] K. Sycara, S. F. Roth, N. Sadeh, and M. S. Fox. Distributed constrained heuristic search. *IEEE Transactions on Systems, Man, and Cybernetics*, 21(6):1446–1461, December 1991.
- [11] 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*, pages 88–102, 1995.
- [12] M. Yokoo and E. H. Durfee. Distributed constraint satisfaction for formalizing distributed problem solving. In *12th IEEE International Conference on Distributed Computing Systems*, pages 614–621, 1992.
- [13] R. Zivan and A. Meisels. Synchronous vs asynchronous search on DisCSPs. In *Proceedings of the First European Workshop on Multi-Agent Systems (EUMA)*, 2003.
- [14] R. Zivan and A. Meisels. Dynamic ordering for asynchronous backtracking on discsps. In *Proceedings of CP'05*, 2005.