Enforcing Soft Arc Consistency on DCOPs with Multiple Variables per Agent

Jimmy H. M. Lee¹, Pedro Meseguer², Wen Su¹

¹ The Chinese University of Hong Kong, Shatin, N.T., Hong Kong
{jlee,wsu}@cse.cuhk.edu.hk

² IIIA - CSIC, Campus Universitat Autònoma de Barcelona, 08193 Bellaterra, Spain
pedro@iiia.csic.es

Abstract. While most of search-based algorithms for optimal DCOP solving assume a single variable per agent, many DCOP instances could be more appropriately modeled with several variables per agent. There are several approaches to transform a DCOP instance with several variables per agent into another instance with one variable per agent, on which existing algorithms could be applied. We present a hybrid approach that combines two of these transformations. Interestingly, our method can be connected with enforcing soft arc consistency during search, a technique that has been shown beneficial for search-based DCOP algorithms. Using BnB-ADOPT⁺ as the solving algorithm, preliminary experimental results on DCOP random instances indicate that this hybrid approach provides clear advantages over the two transformation approaches taken in isolation, and confirm also the benefits of soft arc consistency enforcement in this context.

1 Introduction

Distributed Constraint Optimization Problem (DCOP) can model a wide variety of multiagent coordination problems such as distributed planning [22], distributed scheduling [16], distributed resource and task allocation [24], satellite constellations [2], disaster rescue [14], multiagent teamwork [26], human/agent organizations [5], intelligent forces [4], distributed and reconfigurable robots [25] and sensor networks [27]. They are just a few examples of multiagent applications. DCOP provides a useful framework to investigate how agents can coordinate their decision-making in such domains [20]. In the literature, each agent is often assumed to control only one variable. In some problems, however, each agent could have complex local problems so that they could be more appropriately modeled using DCOPs with multiple variables per agent.

Most existing optimal DCOP solving algorithms are designed with the assumption that each agent handles one variable only, so that there is a reality gap. Problems with complex local problems are usually transformed so that existing algorithms could be applied. Two folklore approaches have been used to achieve the transformation [3]:

– Decomposition or virtual agents (VA): creating a multiple virtual agent for each variable inside each real agent. A different original variable is assigned to each virtual agent, so that this formulation satisfies the one variable per agent assumption. The solving algorithm is executed on each virtual agent, while intra-agent messages are only simulated and often discounted in the calculation of computation cost.
Compilation: converting a DCOP with multiple variables per agent to a DCOP with only one variable by defining a new variable for each agent, whose domain is the Cartesian product of the domains of the original variables assigned to that agent. By construction, there is one new variable per agent, so existing solving algorithms can be applied. Since this technique is applied as a preprocessing step, we call it pre-compilation (PC).

In this paper, we focus on these transformations, and consider their combination in a hybrid approach.

Another focus of our work is with soft arc consistency. Since the late 90s [13, 12], a number of distributed algorithms for optimal DCOP solving have been proposed, such as SBB [13], ADOPT [20], NCBB [6], DPOP [23], AFB [8], BnB-ADOPT [28] and others. In particular, some of those algorithms that are search-based, have been combined with some kind of soft arc consistency (AC) enforcement [19, 10, 9]. Interestingly, our proposal can be easily combined with soft AC enforcement well to obtain symbiotic benefits. Preliminary experimental results confirm some of our intuitions on this kind of problems: enforcing soft AC is useful; the chosen approach (VA or PC) could make a notable difference in the performance of the solving algorithm, while the hybrid approach offers a good and stable performance.

The structure of this paper is as follows. In Section 2 we summarize the main concepts used in the paper. In Section 3 we detail different approaches to deal with several variables per agent, when the solving algorithms at hand assume one variable per agent. In Section 4 we describe our hybrid proposal. In Section 5 we present some results of the proposed approach on random DCOP instances. Section 6 gives concluding remarks.

2 Background

2.1 Formalization

Given a DCOP \( P = (A, X, D, C)^{3} \):

- \( A = \{a^1, ..., a^n\} \) is a set of agents;
- \( X = \{x^1_1, ..., x^1_k_1, x^2_1, ..., x^2_k_2, ..., x^n_1, ..., x^n_k_n\} \) is a set of variables, variable \( x^i_j \) belongs to agent \( a^i \); the set of variables in agent \( a^i \) is \( X^i = \{x^i_1, x^i_2, ..., x^i_k_i\} \);
- \( D = \{D^1_1, ..., D^1_k_1, D^2_1, ..., D^2_k_2, ..., D^n_1, ..., D^n_k_n\} \) is the set of domains for the variables;
- \( C = \{C_{pq}^{ij} | \text{involving different variables } x^i_p \text{ and } x^j_q \} \) is a set of binary functions.

When \( i = j \wedge p = q \), we use \( C^i_p \) to represent the unary cost function.

An optimal solution is a global assignment giving values to all variables with minimum cost. A local assignment \( l^i \) for agent \( a^i \) gives value to the variables of that agent: \( l^i = (\langle x^i_1, v^i_1 \rangle, \langle x^i_2, v^i_2 \rangle, ..., \langle x^i_k_i, v^i_k_i \rangle) \), where \( x^i_k_i \in X^i \) and \( v^i_k_i \in D^i_k_i \). \( L^i \) is the set of all the possible local assignments in agent \( a^i \), \( |L^i| = \prod_{\text{m}} |D^i_m| \). A local assignment is also called a local solution for that agent. Cost\((l^i)\) is the summation of cost functions that only involve variables belonging to agent \( a^i \).

\(^{3}\) The mapping between variables and agents is included in \( X \).
2.2 BnB-ADOPT and Variant

BnB-ADOPT [28] is based on the one variable per agent assumption, and is a memory-bounded asynchronous search algorithm for optimal DCOP solving. It uses the same messages passing and communication framework as ADOPT, but changes the search strategy from best-first search to branch-and-bound depth-first search. BnB-ADOPT prioritizes agents into a pseudo-tree [7] and utilizes AND/OR search tree [21, 18]. Given the toy DCOP shown in Figure 1(a), formed by four binary variables connected by four constraints, Figure 1(b) is one possible pseudo tree for that problem. The dashed edges are “back edges”. Two agents connected by a back edge form pseudo-child pseudo-parent relationship. The corresponding AND/OR search tree appears in Figure 1(c).

BnB-ADOPT uses branch-and-bound search. Its operations aim at reducing the difference between the upper bound and the lower bound at each node. In particular, if \( r \) is the root node of the pseudo tree, BnB-ADOPT decreases the upper bound \( UB^r \), while increases the lower bound, \( LB^r \), until \( UB^r = LB^r \). This is the minimum (optimum) cost of the DCOP instance. This is done though an elaborated reasoning, implemented via message passing. BnB-ADOPT is correct and complete, terminating with the minimum cost [28].

BnB-ADOPT uses three types of messages: VALUE, COST and TERMINATE. VALUE messages contain the variable assignment information of ancestors. After taking value, each agent sends down the pseudotree a VALUE message to all its children and pseudochildren (leaf agents, which have no children, do not send any VALUE). COST messages include aggregate costs and bounds of descendant agents. They go up in the pseudotree, from each agent to its parent (the root agent, which has no parent, does not send any COST). TERMINATE messages are sent from each agent to its children, to terminate search. The basic message flow is shown in Figure 1.

BnB-ADOPT \(^+\) [11] is a new version of BnB-ADOPT that prevents most redundant messages from sending. Each agent keeps the last VALUE message sent to each child/pseudochild and the last COST message sent to its parent. When it has to sent a new VALUE/COST message, it checks whether it is equal to the last VALUE/COST message sent for that destination (unless there is a context change for COST, or the last COST received from that child contains a new boolean field \( ThReq \) equal to true; see [11] for details).
2.3 Soft Arc Consistencies

Enforcing consistencies can prune the suboptimal values that are not involved in the optimal solution, so as to reduce the search space. Soft arc consistencies are enforced between variables. Given a DCOP $P$, let $(i, a)$ represents $x_i$ taking assignment $a$, $C_{ij}$ is the binary cost function between $x_i$ and $x_j$, $C_i$ is the unary cost function on $x_i$, $\top$ is the upper bound and $C_{\phi}$ is the lower bound for $P$. We consider the following soft consistencies defined in [15]:

- **Node Consistency (NC):** $(i, a)$ is NC if $C_{\phi} + C_i(a) < \top$; $x_i$ is NC if all its values are NC and $\exists b \in D_i$ s.t. $C_i(b) = 0$. $P$ is NC if every variable is NC.

- **Arc Consistency (AC):** $(i, a)$ is AC w.r.t. $C_{ij}$ if $\exists b \in D_j$ s.t. $C_{ij}(a, b) = 0$; $b$ is a support of $a$; $x_i$ is AC if all its values are AC w.r.t. every binary cost function involving $x_i$; $P$ is AC if every variable is AC and NC.

AC can be reached by projecting the minimum cost from its binary cost functions to its unary cost, and then projecting the minimum unary cost into $C_{\phi}$. When a value is deleted from the domain of $x_i$, we need to recheck AC on every variable that is constrained with $x_i$. Since the deleted value may be the only support for a variable $x_j$, this deletion could cause further deletions in $x_j$. The AC check must be done until no further values are deleted.

2.4 BnB-ADOPT$^+$ and Soft Arc Consistencies

Gutierrez and Meseguer [10] combined BnB-ADOPT$^+$ and soft arc consistency (soft AC) enforcement considering unconditional deletions only. A deletion is unconditional if this is independent to any value assignment. When combining soft AC and BnB-ADOPT$^+$ (BnB-ADOPT$^+$-AC-UNDO), several modifications to the original algorithm are needed. Regarding messages, a new message DEL is added to inform of value deletions, while an enlarged structure for VALUE and COST messages includes elements needed for soft AC enforcement. Regarding computation, each agent holds one copy of constrained agents’ domains and related binary cost functions for consistency enforcement. Only the agent owner of a variable could modify its domain.

When enforcing soft AC in a distributed environment, simultaneous deletions may happen which could cause two agents maintain different copies of the same cost function. To tackle that problem, Gutierrez and Meseguer [10] proposed to synchronize the deletions between two constrained agents. However, synchronizing deletions not only introduces extra messages, but also slow down the consistency enforcement and even the search process. Gutierrez et al. [9] devised a more efficient way to keep cost function copies identical, which is called undo mechanism. Observing that different copies of cost functions result from different orderings of operations being enforced in different agents, undo mechanism is a way to avoid that.

Enforcing soft AC during BnB-ADOPT$^+$ search has been shown very beneficial for performance [10, 9], substantially decreasing the communication and computation required to solve DCOP instances to optimality. It is worth noting that enforcing soft AC approach has been developed under the assumption of one variable per agent (as most existing algorithms for DCOPs).
3 Multiple Variables per Agent

When more than one variable is assigned to the same agent, the common assumption of one variable per agent no longer holds. However, existing solving algorithms—developed under this assumption—can be useful for solving these DCOPs. In the following, we detail the two approaches mentioned in Section 1 that can be directly applied to handle this issue, namely decomposition (also called virtual agents) and pre-compilation.

3.1 Virtual Agents

The decomposition—also called virtual agents (VA)—approach considers producing from the original DCOP $P$ a new DCOP $P' = (A', X', D', C')$ such that it satisfies the assumption of one variable per agent, so that existing solving algorithms can be applied to it [3]. The VA approach is as follows. The set of variables, domains and constraints in $P'$ are the same as those in the original $P$. However, $A'$ is different from $A$: there is a different (virtual) agent per original variable (to which it is assigned). Now, the new $P'$ satisfies the one variable per (virtual) agent assumption. Considering a real agent as a single CPU, the agent would contain as many virtual agents (processes) as variables are assigned to it. Each virtual agent holds a single variable and executes a copy of the solving algorithm. Communications between virtual agents of the same real agent is done by shared memory, while virtual agents belonging to different real agents communicate via network.

One of the advantages of using this approach is its low implementation efforts requirement. Given any solving algorithm, we could directly apply this approach on it. The other is that it maintains the original constraint graph on variables, which is beneficial when trying to divide the whole problem into independent parts. As described before, BnB-ADOPT*-AC-UNDO is based on the pseudo-tree structure. Value assignments in different branches in a pseudo-tree could be done concurrently and independently. Compared with compilation methods, which build a pseudo-tree on agents, the VA approach could capture more concurrency gain from the pseudo-tree structure, especially when the connectivity is not dense.

However, this approach does not provide a global view of a real agent. There are messages sent from one real agent to another, but by different virtual agents. These messages could be conveyed in one message from one real agent to another. When the connectivity is getting denser, the opportunity of sending messages to the same real agent by different virtual agents in the same real agents becomes larger. Hence, more messages could be aggregated into one.

3.2 Pre-compilation

The pre-compilation (PC) approach considers producing from the original DCOP $P$ a new DCOP $P'' = (A'', X'', D'', C'')$ such that it satisfies the assumption of one variable per agent, so that existing solving algorithms can be applied to it [3]. Instead of generating as many (virtual) agents as original variables (the VA approach described in the previous Section), this approach considers generating as many new variables—called
macro variables– as real agents in the set $A$. We define for each agent $a^i$ a macro variable $z^i$, whose domain $D_i = \prod_{j=1}^{k_i} D_j^i$ is the Cartesian product of the domains of the original variables handled by that agent. On these new variables, the following new constraints are defined. For each agent $a^i$, we add a unary cost function $f^i = \sum_{p,q=1}^{k_i} C_{ij}^{ii}$, that is made from the original constraints between original variables that are all inside agent $a^i$. For each pair of constrained agents $a^i$ and $a^j$, we add a binary cost function $f^{ij} = \sum_{p,q=1}^{k_i,k_j} C_{ij, i \neq j}^{ij}$, that is made from the original constraints between original variables of agent $a^i$ and agent $a^j$, $i \neq j$.

It is clear that $P''$ satisfies the one variable per agent assumption: one copy is executed per each real agent, now holding a single macro variable. This transformation is rather direct and does not present any conceptual difficulty. However, it has a drawback: the domain size of each macro variable is exponentially large with respect to the domains of the original variables. This may cause memory problems when this approach is implemented in practice.\footnote{A similar situation happened in the past, when non-binary CSPs were transformed in binary ones by the dual graph approach: new variables suffered from exponentially large domains [1].}

### 3.3 Improved Pre-compilation

In the context of compilation, in [3] authors propose an improved compilation approach. They divide original variables in two groups: external variables, which are those variables that are constrained with at least one variable in other agents, and private variables, those exclusively constrained with variables inside the same agent. Figure 2 shows a constraint graph for a DCOP with multiple variables per agent. Each oval represents an agent and the circles inside each agent represent the variables belonging to that agent. Variables $x^3_1$ in agent $a^3$, $x^4_0$ and $x^4_1$ in agent $a^4$ are the private variables of this example, while the others are external.

![Fig. 2.](image-url)

According to [3], any local solutions that have identical assignments to those external variables are equivalent with respect to the distributed problem. If there is more than one optimal local solution with the same assignments to external variables, the solutions are fully interchangeable, and so only one is required. Local solutions with
identically assigned external variables but sub-optimally assigned other variables are strictly dominated. These sub-optimal local solutions are thus discarded when adopting improved compilation. The domain of a macro variable could be reduced from the Cartesian product of all the variables to that of only external variables.

Let us consider again the example of Figure 2. Given two local solutions for agent $a^4$: $l^4_i = (\langle x^4_0, t_i \rangle, \langle x^4_1, u_i \rangle, \langle x^4_2, v \rangle, \langle x^4_3, w \rangle)$ and $l^4_j = (\langle x^4_0, t_j \rangle, \langle x^4_1, u_j \rangle, \langle x^4_2, v \rangle, \langle x^4_3, w \rangle)$, where external variables are assigned the same values. If $\text{Cost}(l^4_i) > \text{Cost}(l^4_j)$, the local solution $l^4_i$ is strictly dominated by $l^4_j$. Therefore, $l^4_i$ could be removed from the domain of agent $a^4$ when compiling agent $a^4$. As a result, the domain size of macro variable $z^4$ is reduced from $|D^4_0| \times |D^4_1| \times |D^4_2| \times |D^4_3|$ to $|D^4_2| \times |D^4_3|$. When implementing PC, we use improved compilation [3], which reduces the domain of macro variables. Although reduced, the domain size of the macro variables could be exponentially large. This remains as the main drawback of the pre-compilation approach.

Using this approach, a message from agent $a^i$ contains the information of all the external variables in $a^i$. When inter-agent connectivity is dense, external variables dominate the overall variables in an agent. So it is not practical to compile the variables in an agent into one macro variable when the number of external variables is large. Another issue is that it loses some opportunities to perform search concurrently. Building a pseudotree on real agents conceals the detailed constraints between variables. Especially when the connectivity between variables are not dense, we could decompose the problem to more independent subproblems using the VA approach.

### 3.4 Enforcing Soft AC

Any of the previous approaches can be combined with enforcing soft AC. Since they produce a DCOP $P'$ or $P''$ that satisfies the one variable per agent assumption, existing methods for enforcing soft AC in DCOP solving [10, 9] can be directly applied. Generally speaking, enforcing soft AC increases the performance of BnB-ADOPT$, combination with the above mentioned approaches for DCOPs with multiple variables per agent will also enhance their performance.

### 4 Hybridizing VA and PC approaches

Using the VA approach, we could benefit from its high concurrency, while losing some ability to aggregate messages when they share the same source and destination agent. In contrast, the PC approach provides a way to aggregate messages, however, with weaker concurrent capability. Also, the PC approach suffers from the exponentially large memory requirement. Based on these observations, we propose to hybridize these two approaches, as we explain next. Our proposal is passing from a DCOP $P$ with several variables per agent to another $P'''$ such that for some agents the transformation follows the VA approach, while for others it follows the PC approach. This hybrid approach is motivated by the following reasons:

1. **Hybridize by Domain Size.** The idea is that we only do compilation for an agent when the domain size of resulting macro variable is below a certain threshold. If an
agent is not chosen to be compiled, we use the VA approach for this agent. In this way, we limit the required memory, assuring that the PC approach is applied only when the practical conditions allow so.

2. HYBRIDIZE BY CONNECTIVITY. The idea is that we follow the PC approach for an agent when its intra-agent connectivity is larger than certain threshold. For agents with lower intra-agent connectivity, we use the VA approach. Observe that when local problems are dense, it is more likely that variables would belong to the same branch in the pseudotree on variables. Or, in other words, there are less options for concurrency (and, in this case we could benefit less from the VA approach). On the other hand, variables in the same branch may send messages to the same destination, which indicates that some messages could be aggregated. Therefore, we propose to compile strongly constrained variables in an agent into one macro variable.

3. HYBRIDIZE BY BOTH DOMAIN SIZE AND CONNECTIVITY. This approach limits the agents that could be compiled into macro variables more strictly by combining the above two conditions. Only if an agent will result in a macro variable whose domain size is smaller than a certain threshold and also, its intra-agent connectivity is larger than a certain threshold, we do compilation for this agent. If not, we create a virtual agent for each variable inside this agent.

The approach can be seen as a combination of the VA and PC approaches. From $P$, the VA approach changes its set of agents, keeping untouched the original variables, domains and constraints. On the other hand, the PC approach maintains the original set of agents, changing the set of variables, domains and constraints. The hybrid approach changes all sets, on a real agent basis: some real agents are treated by the VA approach while others are treated by the PC approach. Variables, domains and constraints change accordingly. The resulting instance –that it would not be produced by the VA nor the PC approaches executed in isolation– satisfies the one variable per agent (now a mixture of virtual and real agents) assumption, so existing solving algorithms can be applied to it. In addition, it directly allows to include soft AC enforcing.

5 Empirical Evaluation

We evaluate the performance of different approaches for DCOPs with multiple variables per agent, namely virtual agents (VA) and pre-compilation (PC), with the hybrid approaches described in the previous Section: hybrid by domain size (HD), hybrid by connectivity (HC), hybrid by both domain size and connectivity (HDC). We set the threshold of HD approach to be 100, i.e., we do compilation for an agent if the Cartesian product of its external variables is less than or equal to 100. For other agents, we create a virtual agent for each variable inside those agents. The threshold of HC approach is set to be 0.5. Given an agent holding $n$ variables, when the number of constraints between variables inside this agent is larger than $0.5 \times n \times (n - 1)/2$, we compile the variables into a macro variable using improved compilation. Or else, we treat each variable in the agent as a virtual agent. As to the HDC approach, we choose the agents that not only hold the condition to be compiled by HD approach, but also satisfy the condition of HC approach, to be compiled.
As solving algorithms, we use BnB-ADOPT and BnB-ADOPT+.-AC-UNDO, comparing their performance. While the former solves the problem by search only (branch-and-bound), the latter also include enforcing soft AC during search.

We simulate the DCOP solving process on AgentZero framework [17], which is designed for multi-agent oriented tasks. Agents act asynchronously using multi-threads techniques. We evaluate the performance in terms of communication cost (total number of messages exchanged among agents) and computation effort (non-concurrent constraint checks, NCCCs). Message delay is added while executing the algorithms. We use asynchronous random message delay (AMDS) proposed in [29]. We set the random delay range to be $[0, 100]$ clock ticks.

We test our approaches on binary random DCOPs [10]. All the problems are characterized by $\langle a, n, d, p, q, T \rangle$, where $a$ is the number of agents, $n$ is the number of variables, $d$ is the domain size of each variable, $p$ is the number of intra-agent constraints, $q$ is the number of inter-agent constraints and $T$ is the distribution of variables into agents. The distribution of cost functions follows [10]. Intra-agent constraints are scattered into different agents randomly, while the total number of intra-agent constraints are specified by parameter $p$.

We evaluate our approaches on following two batches of random DCOP instances:

- Batch1: evenly distributed random DCOPs with $\langle a = 4, n = 16, d = 3, p \in \{4, 8, 16, 24\}, q \in \{3, 4, 8, 12\}, T = (4, 4, 4, 4) \rangle$. Each problem in this batch have 4 variables
Fig. 4. Number of solved instances within the limits explained in the text, in batch2.

Fig. 5. Random DCOPs with unevenly distributed variables in agents (batch2).
per agent. We run 50 instances for each parameter setting. Results are reported in Figure 3.

- Batch2: not evenly distributed random DCOPs with \( \langle a = 5, n = 30, d = 3, p \in \{9, 12, 26, 37, 46, 74\}, q \in \{4, 8, 12\}, T = (2, 4, 6, 8, 10) \rangle \). There are 2, 4, 6, 8, 10 variables in different agents. We run 20 instances for each parameter setting. Some problems in this batch are too hard to be solved in a reasonable time, so we included a timeout in wall-clock time and a limit regarding the number of exchanged messages. If a problem is not solved within half an hour or has exchanged more than \( 5 \times 10^5 \) messages, we will terminate the execution. The number of instances that are solved within these limited time and number of messages appears in Figure 4. And the results of the number of messages and NCCCs are shown in Figure 5.

From this results (that we consider preliminary given the number of instances tested) we observe the following facts. Enforcing soft AC improves performance. This can be clearly seen from Figures 3, 4 and 5. This fact is not new: it is just a confirmation of something already observed in the past for DCOP solving. Our hybrid approaches get the best of the VA and PC approaches, both in number of messages and NCCCs. While the hybrids offer the best performance overall, forming the option of choice, it is difficult to identify the best performer among the different versions tested. More experimental work is needed to answer this point, although the hybrid proposals appears to be promising for solving DCOP with multiple variables per agent efficiently.

6 Conclusion

We presented a hybrid approach for dealing with DCOP with multiple variables per agent. While the classical approaches, VA or PC, consider to apply them monolithically on a DCOP instance, we propose to combine them, motivated by several reasons (exponentiality, concurrency). Thus, a transformed instance may have some real agents transformed by the PC approach, while others are dealt by the VA approach. \(^5\) We have considered several criteria to apply this hybrid approach. Experimentally, the hybrid approach seems to perform better than each of the above mentioned approaches in isolation, over a random set of DCOP instances. It can be easily combined with soft AC enforcing, which enhances its performance, while keeping its correctness.

Acknowledgments

We thank the anonymous reviewers for their constructive comments, which helped much to improve the paper. This research has also been substantially supported by the following grants: CUHK413808, CUHK413710 and CUHK413713 from the Research Grants Council of Hong Kong SAR, the CSIC/RGC Joint Research Scheme grants S-HK003/12 and 2011HK0017, TIN2013-45732-C4-4-P from Spanish MINECO, and Generalitat de Catalunya SGR-2014-118.

\(^5\) Considering each real agent as a single CPU, an agent may thus execute several coexisting solving processes (each associated with a virtual agent) or simply one single process (for dealing with a macro variable).
References