Resilient Distributed Constraint Optimization Problems

Yarden Naveh[1], Roie Zivan[1], and William Yeoh[2]

[1] Department of Industrial Engineering and Management, Ben Gurion University of the Negev
[2] Department of Computer Science, New Mexico State University
{navehya,zivanr}@bgu.ac.il wyeoh@cs.nmsu.edu

Abstract. Dynamic distributed constraint optimization problems (Dynamic DCOPs) are useful in modeling various distributed combinatorial optimization problems that are dynamically changing over time. Previous attempts assume that it is possible for agents to take on a new solution each time the problem changes. However, in some applications, it is necessary to commit to a *single* solution at the start of the problem regardless of how the problem may change in the future. In this paper, we make the following contributions: (*i*) We propose a *Resilient DCOP* (R_DCOP) model, which is a dynamic DCOP but the goal is to find a single solution that is *resilient* to future changes of the problem; (*ii*) We introduce a naive complete algorithm to solve this problem as well as several enhancement methods that can be used to speed it up; (*iii*) We provide theoretical analyses on the complexity of this problem and of our algorithm; and (*iv*) We empirically demonstrate the feasibility of our algorithm to solve R_DCOPs.

1 Introduction

There is a growing need for optimization methods to support decentralized decision-making in complex multi-agent systems. Consider a disaster rescue scenario, where rescue units (medical personnel, fire fighters, police) need to coordinate their actions so as to save as many victims as possible. A promising multi-agent approach to solve these types of problems is to model them as *distributed constraint optimization problems* (DCOPs) [14], whose goal is to optimize a global objective in a decentralized manner. However, it assumes that the problem does not change over time. Previous attempts to cope with dynamism in DCOPs have focused on modeling a *Dynamic DCOP* as a sequence of (static) DCOPs and *reactively* solving the new DCOP after a change occurs [12, 20, 23]. More recently, researchers have proposed offline *proactive* approaches that anticipate possible future changes and take them into account when finding Dynamic DCOP solutions.

These existing methods make the following key assumption: Agents can change their solutions each time the problem changes. In reactive approaches, the agents collectively react to find a new solution for the current DCOP; in proactive approaches, the agents take into account possible future changes to the problem and collectively plan their sequence of solutions, one solution in each future time step of the problem. However, there exists multi-agent applications where this assumption is invalid. Taking our example disaster rescue scenario, the allocation of rescue units throughout the country (e.g., the number of police officers that are assigned to the various police stations throughout the country) must be committed for a reasonably long period of time (e.g., a year) before they are revisited again. However, the underlying problem (e.g., the number of crimes committed) may change very regularly (e.g., hourly).

Therefore, in this paper, we pursue an innovative direction that is orthogonal to the existing work in Dynamic DCOPs. We propose the *Resilient DCOP* (*R_DCOP*) model, which, like a dynamic DCOP, is also modeled as a sequence of (static) DCOPs. However, the objective is to find a *single* DCOP solution that does not change for a majority of consecutive time steps in the problem. However, if necessary, it can be changed if the resulting change will significantly improve the quality of the solution. Using our example above, the allocation of police officers will remain unchanged for a long period of time but can be periodically revisited if a new allocation will significantly decrease the number of crimes committed. An R_DCOP solution must be *resilient* by considering all possible future changes in the problem. It will either maintain a high quality despite future changes or can be easily adapted if necessary. We measure the quality of a solution through a balanced combination of the current cost (with respect to the current DCOP) and future expected costs (with respect to possible future DCOPs) of the problem, and we measure the ease of adaptation in terms of similarity to a previous solution (e.g., the number of perturbations to the old solution required to arrive at the new solution). The use of similarity is a good fit in many practical applications. Using our example again, one should aim to minimize the number of police officers that need to be transferred from one police station to another as such transfers are often inconvenient to the officers and have large impacts on their life.

To solve our R_DCOP model, we introduce a naive complete algorithm for solving it and propose combinations of methods that can be used to enhance the algorithm. We also provide some theoretical analysis on the complexity of this new R_DCOP model as well as the complexity of our algorithm. Finally, our empirical evaluations demonstrate the impact of the various enhancement methods on the baseline naive algorithm as well as sensitivity analyses for how parameters of the problem (e.g., horizon, discount factor) affects the algorithm.

2 Distributed Constraint Optimization Problem

A *DCOP* is a tuple $\langle \mathcal{A}, \mathcal{X}, \mathcal{D}, \mathcal{R} \rangle$. $\mathcal{A}$ is a finite set of agents $\{A_1, A_2, \ldots, A_n\}$. $\mathcal{X}$ is a finite set of variables $\{X_1,X_2,\ldots,X_m\}$. Each variable is held by a single agent (an agent may hold more than one variable). $\mathcal{D}$ is a set of domains $\{D_1,\ D_2,\ldots,D_m\}$. Each domain D_i contains the finite set of values that can be assigned to variable X_i. We denote an assignment of value $d \in D_i$ to X_i by an ordered pair $\langle X_i, d \rangle$. $\mathcal{R}$ is a set of relations (constraints). Each constraint $C \in \mathcal{R}$ defines a non-negative *cost* for every possible value combination of a set of variables, and is of the form $C : D_{i_1} \times D_{i_2} \times \ldots \times D_{i_k} \rightarrow \mathbb{R}^+ \cup \{0\}$. A *binary constraint* refers to exactly two variables and is of the form $C_{ij} : D_i \times D_j \rightarrow \mathbb{R}^+ \cup \{0\}$. A *binary DCOP* is a DCOP in which all constraints are binary. A *partial assignment* (PA) is a set of value assignments to variables, in which each variable appears at most once. *vars(PA)* is the set of all variables that appear in PA, $vars(PA) = \{X_i \mid \exists d \in D_i \wedge \langle X_i, d \rangle \in PA\}$. A constraint $C \in \mathcal{R}$ of the form $C : D_{i_1} \times D_{i_2} \times \ldots \times D_{i_k} \rightarrow \mathbb{R}^+ \cup \{0\}$ is *applicable* to PA if $X_{i_1}, X_{i_2}, \ldots, X_{i_k} \in vars(PA)$. The *cost of a partial assignment* PA is the sum of all applicable constraints to PA over the assignments in PA. A *complete assignment* (or a *solution*) is a partial assignment that includes all the DCOP's variables ($vars(PA) = \mathcal{X}$). An *optimal solution* is a complete assignment with minimal cost. For simplicity, we make the standard assumptions that all DCOPs are binary and that each agent holds exactly one variable.

3 Related Work

A *Dynamic DCOP* is typically modeled as a sequence of static DCOPs with changes between them. There are two general lines of research: (1) *Reactive* algorithms are usually online and wait for changes to take place before finding a solution to the new problem [17, 20, 23]. Variations have considered agents that incur costs for changing their value assignments when the problem changes and deadlines for choosing new values [18], agents with imperfect knowledge about their environment [11], and the specific needs of mobile sensing teams [24]. These reactive approaches do not explicitly model possible future changes and, thus, differ from our approach. (2) *Proactive* algorithms are usually offline and finds a solution for each future time step of the problem [8, 9]. These algorithms do model possible future changes and do take them into account when searching for their solutions. However, as the solution for each time step may be different, they differ from our approach that seeks to find a single resilient solution.

Within centralized constraint reasoning models, knowledge of uncertain information was modeled and combined in solvers of constraint satisfaction problems (CSPs) such as *Mixed CSPs* [7] and *Stochastic CSPs* [22, 21]. To the best of our knowledge, the approaches proposed for CSPs in these papers were not applied to dynamic distributed optimization problems. Nevertheless, our work is inspired by the methods proposed in these studies.

Within the planning literature, there are models that bear similarity to R_DCOPs and Dynamic DCOPs in general. The first is *Markov decision processes* (MDPs) and *partially-observable MDPs* (POMDPs) [10], which, like R_DCOPs, also model possible future changes to the problem. However, they are centralized models for single-agent problems. The closest form of POMDPs to R_DCOPs is Online POMDP [19], where, similar to reactive Dynamic DCOP algorithms, a solution for each time step is computed and then executed, before the solution for the next time step is computed. This differs from R_DCOPs where the objective is to find a single resilient solution. There exist also decentralized extensions of MDPs and POMDPs, called Dec-MDPs [2, 1, 5, 6] and Dec-POMDPs [16, 4]. One model even incorporated POMDPs and DCOPs into a single model, the *networked distributed POMDP* (ND-POMDP) [15]. These models are more general then the proposed R_DCOP model as they can represent sequential execution similar to proactive Dynamic DCOP methods. Again, this differs from R_DCOPs where the objective is to find a single resilient solution.

While we could have presented R_DCOPs as an extension of DCOPs or as a restricted form of Dec-MDPs or POMDPs, we chose to present it as a DCOP extension as our techniques and algorithms use distributed DCOP search algorithms as building blocks. Regardless, this presentation choice should not affect the novelty and significance of the proposed model and algorithm.

4 Resilient DCOP (R_DCOP)

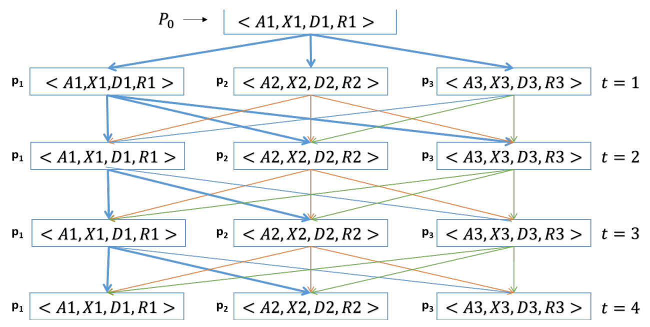

Fig. 1: R_DCOP Example

An R_DCOP is a tuple $\langle P_0, H, E, S, \Pi, \gamma_0, \gamma_- \rangle$. P_0 is the initial DCOP, which we aim to solve. H represents the time horizon, where, at each time step $1 \leq t \leq H$, possible changes can occur. $E = \{E_0, \ldots, E_k\}$ is a set of dynamic elements, where, for each $E_i \in E$, there exists a finite discrete domain of possible states $S_i = \{S_{i1}, \ldots, S_{iq}\} \in S$ that it can be in. The set of all possible global dynamic states, i.e., combinations of the states that all dynamic elements can be in, is denoted by $\Delta = S_1 \times \ldots \times S_k$ and $\Pi : \Delta \to [0, 1]$ is a function assigning a probability $\Pi(\delta)$ to each global dynamic state $\delta \in \Delta$. The goal is to find a solution to P_0 that optimizes the current derived cost and expected future cost with respect to similarity consequences.

Figure 1 illustrates an example of an R_DCOP. In this example, $|\delta| = 3$ as there are 3 possible subproblems (DCOPs) at each time step $1 \leq t \leq 4 = H$. The subproblems on the left are represented by the tuple $\langle A1, X1, D1, R1 \rangle$ and they can occur with probability p_1 in all time steps. This subproblem is also the main problem P_0 for which we are seeking a *resilient* solution. The middle subproblems are represented by $\langle A2, X2, D2, R2 \rangle$, which can occur with probability p_2 and the right subproblems are represented by $\langle A3, X3, D3, R3 \rangle$, which can occur with probability p_3.

Dissimilarity between various solutions at different time steps is penalized in order to make sure changes occur only when they are truly beneficial in spite of their imposed cost. Two types of penalties are taken into account: γ_- represents the penalty for a change between two consecutive time steps, and γ_0 represents the penalty for a change between the solution to P_0 and the solution being examined for a problrm in a later time step. This separation for two types of penalties models real life scenarios where long term decisions that are made in advance may be less flexible to change than ongoing changes. The expected cost of a solution ϕ_t to problem P given the previous solution ϕ_{t-1} and the initial solution ϕ_0 is computed as follows:

$$C_t(\phi, P \mid \phi_{t-1}, \phi_0) \tag{1}$$

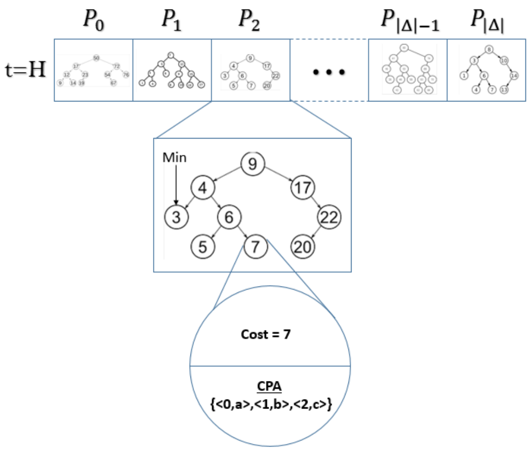

Fig. 2: Solutions Data Structure at $t = H$

$$= \begin{cases} F(\phi, P) + \gamma(\phi, \phi_{t-1}, \phi_0) \\ \qquad = F(\phi, P) + \gamma_{-}(\phi, \phi_{t-1}) + \gamma_0(\phi, \phi_0) & t = H \\ F(\phi, P) + \gamma(\phi, \phi_{t-1}, \phi_0) \\ \qquad + \sum\limits_{\delta \in \Delta} \Pi(\delta) \min\limits_{\phi'} \{C_{t+1}(\phi', T(P, \delta) | \, \phi, \phi_0)\} & 1 \leq t \leq H \\ F(\phi, P) + \sum\limits_{\delta \in \Delta} \Pi(\delta) \min\limits_{\phi'} \{C_1(\phi', T(P, \delta) | \, \phi, \phi_0)\} & t = 0 \end{cases}$$

where $F(\phi_t, P)$ is the cost of solution ϕ_t on the subproblem P, the γ function is the summation of γ_0 and $\gamma_{\_}$ and $T(P, \delta)$ is the problem that results from applying the dynamic events represented by δ to P. The objective is to find the complete assignment (optimal solution) ϕ_0^* such that:

$$\phi_0^* = \underset{\phi_0}{\operatorname{argmin}} \{C_0(\phi_0, P_0 | \phi_0, \phi_0)\} \tag{2}$$

5 Naive Complete Algorithm

We propose a synchronous naive complete R_DCOP algorithm that traverses all possible assignments of all possible subproblems (DCOPs) at each time step. Calculating the expected cost at a specific point in time requires information regarding the possible costs that future changes can yield. Therefore, a natural course of action would be to calculate expected costs from the last time step and proceed backwards recursively. For this bottom-up approach, a dynamic programming data structure, which will be managed by the last agent,[3] will be used to store relevant data and calculate costs between two consecutive time steps. Figure 2 demonstrates this data structure maintained at time step $t = H$. For each subproblem, a binary tree is maintained, holding the solutions of

[3] We leave the task of designing a distributed version of this data structure for future work.

Procedure Start

```
1  if is first agent then
2      $cpa \leftarrow initCPA$;
3      $pc \leftarrow \langle t = 0;\ \delta = P_0;\ \phi_0 = cpa \rangle$;
4      assignCPA($\phi_0$, pc);
```

Procedure assignCPA(cpa,pc)

```
5  if agent exists in pc.$\delta$ then
6      if current domain is not empty then
7          assign minimal cost value to cpa;
8          passCPA(cpa, pc);
9      else
10         backtrack(cpa, pc);
11 else
12     passCPA(cpa, pc);
```

that subproblem sorted by their costs in order to enable efficient memory pruning in our enhanced version (as will be described in the next section). For a better understanding of the algorithm, we first present a general outline and then present the actual procedures performed by the agents:

- The agents systematically traverse all possible solutions (ϕ_0) for the initial subproblem P_0.
- For each such solution ϕ_0, for each time step $t = H, H-1, H-2, \ldots, 1$, agents calculate and store for every possible subproblem (at this time step t), the cost of every possible solution, taking into account the stored costs for solutions of future time steps ($t' > t$) and the cost of assignment changes. At the end of this step, the cost for ϕ_0 is calculated.
- The solution ϕ_0^* for the R_DCOP is the assignment to P_0 for which a minimal cost was calculated.

The code for the main functions of the algorithm is presented next. The first agent starts by initiating the current partial assignment (CPA) message and assigning its variable (line 2 of the pseudo-code). This partial assignment will be extended to a complete assignment ϕ_0 to P_0. In the *assignCPA* procedure, an agent, as part of the attempt to find a solution to a subproblem at some (possibly future) time step, extends the current assignment by assigning its variable (line 7), or backtracks if all its domain has been exhausted (line 10). Since the existence of an agent may be a dynamic element, if the future subproblem examined does not include the agent, it passes on the CPA without extending it (lines 5, 12).

All agents apart from the last agent send a CPA that they were able to extend forward to the next agent (line 20). The last agent in time steps larger than zero stores the solution cost and backtracks (lines 17, 18). If the time step is zero, the last agent needs to initiate the examination of a new solution to P_0 (line 15).

Procedure passCPA(cpa, pc)

```
13 if is last agent then
14     if pc.$t$ = 0 then
15         exploreTimeStep($\phi_0$ =cpa,$t = Horizon$);
16     else
17         submitSolution($\delta$, cpa);
18         backtrack(cpa,pc);
19 else
20     send("CPA", pc, cpa).toNextAgent();
```

Procedure WR: CPA(pc,cpa)

```
21 if agent exists in pc.$\delta$ then
22     initialize current domain;
23     assignCPA(pc,cpa);
24 else
25     passCPA(pc,cpa);
```

In the backtrack procedure, agents either reassign their variables (lines 37, 38) or, if the domain is exhausted or they are not active in this subproblem, pass the CPA backwards (lines 35, 46). If the backtracking agent is the first agent and $t \neq 0$, then it means that a subproblem has finished being examined and *ProblemBT* (problem backtrack) is initiated to continue examining other subproblems (lines 32, 44). Otherwise, it means that all possible solutions for P_0 have been examined and the algorithm terminates (lines 30, 44).

"ProblemBT" messages initiate the change of dynamic elements, which result in the next subproblem to be solved. An agent receiving such a message reassigns its dynamic elements (line 51). If all it's dynamic states are exhausted, then, if it is not the first agent, it sends a "ProblemBT" message to the previous agent (lines 57, 58). If this agent is the first agent, then it means that all subproblems at the current time step t have been examined and the agent initiates the examination of the previous time step $t-1$ (or terminates the algorithm if $t = 0$).

5.1 Complexity

R_DCOPs extend DCOPs with the addition of probabilistic information and similarity requirements. These impose the possible existence of multiple DCOPs in each time step in the horizon and the need to solve them. Thus, the complexity of R_DCOPs is at least as hard as the complexity of standard DCOPs, which is NP-hard.

Computational Time Complexity: The computational time for a naive and exhaustive algorithm is as follows: If $d = \max_{1 \leq i \leq n} |D_i|$ and $s = \max_{1 \leq i \leq k} |S_i|$, then the number of possible dynamic states is bounded from above by s^k and the number of possible assignments is bounded from above by d^n. Thus, computation of the cost C_H (the expected cost at the final time-step H) by exhaustive enumeration of all inputs $\phi_t, \phi_{t-1}, \phi_0, P$ requires $O(\omega d^{3n} s^k)$ steps, where ω is the number of the constraints. A computation time of C_t (for $0 < t < H$) required to perform exhaustive enumeration

```
Procedure backtrack(pc,cpa)
if agent exists in $pc.\delta$ then
    if current domain = $\emptyset$ then
        if is first agent then
            if $pc.t$ = 0 then
                finish();
            else
                send("ProblemBT",pc).toLastAgent();
        else
            unassign(cpa);
            send("BT_CPA",pc,cpa).toPreviousAgent();
    else
        remove current assignment from current domain;
        assignCpa(pc,cpa);
else
    if is first agent then
        if $pc.t = 0$ then
            finish();
        else
            send("ProblemBT",pc).toLastAgent();
    else
        send("BT_CPA",pc,cpa).toPreviousAgent();
```

```
Procedure WR: BT-CPA(pc,cpa)
if is last agent $\wedge t = 0$ then
    submit solution cpa to $P_0$;
backtrack(pc, cpa);
```

of all inputs $\phi_t, \phi_{t-1}, \phi_0, P$ is $O((\omega + d^n s^k)(d^{3n} s^k))$. Finally, computing ϕ_0^* given C_1 requires summing the costs of all ω constraints while solving P_0 in addition to summing the weighted maximum utilities of all s^k problems at time step $t = 1$ for each one of the d^n possible solutions of P_0, thus requiring a total time of $O(d^n(\omega + d^n s^k))$. Thus, the total computational time is the summation of all of the above:

$$O(\omega d^{3n} s^k + (H-1)(\omega + d^n s^k) + d^n(\omega + d^n s^k)) =$$

$$O(\omega d^{3n} s^k + H\omega + H d^{2n} s^k)).$$

Memory Complexity: The naive algorithm uses dynamic programming to calculate the optimal cost C_0. In the worst case, we need to store in memory two complete consecutive layers of time steps, each one maintaining the full assignments (n values) and their costs for all d^n solutions of each of the s^k subproblems. In total, the memory that needs to be stored is: $O(ns^k d^n)$.

Procedure WR: ProblemBT(cppa)

```
if $\delta_i$ is not exhausted then
    $pc.\delta$.assign(nextAssignment);
    if isLastAgent() then
        explore $\delta$;
    else
        send("ProblemPromotion", pc).toNextAgent();
else
    if is not first agent then
        send("ProblemBT").toPreviousAgent();
    else
        timeStepDone(cppa);
```

5.2 Termination

In order to prove the termination of the algorithm, we need to show that for every assignment ϕ_0: 1) No identical assignment of values to variables is ever examined within the same subproblem. 2) No identical subproblem is ever examined within the same time step. 3) No identical ϕ_0 is ever examined as a solution to P_0. The systematic exhaustive nature of the algorithm ensures the above is true. We omit details of the proof due to a lack of space.

6 Enhancement Methods

The naive algorithm presented can be significantly improved by combining a number of search techniques that result in effective pruning of the search space. These techniques build upon existing methods used in static distributed algorithms. However, their implementation in a dynamic scenario allows reasoning across time steps, which prevents redundant duplication of search effort in similar problems in different time steps.

(Method 1) Memory Pruning. This method aims to reduce cross time step checks, which are checks of the cost imposed due to a change of an assignment to a variable between two time steps with respect to the examined solution ϕ_0. The dynamic programming data structure holds the solutions to subproblems and their costs. This method prevents a solution from being stored if the improvement it offers is less than the cost for shifting to it, i.e., it is dominated. The fewer the number of solutions stored at time step t, the fewer the number of comparisons that need to be made at time step $t-1$ and, thus, when a cost for a solution is not stored, the runtime of the algorithm is reduced as well.

(Method 2) Using SBB to Solve Subproblems. While the general design of the algorithm uses dynamic programming, each of the subproblems can be solved using Synchronous Branch and Bound (SBB). Thus, as part of the branch-and-bound procedure in SBB, the algorithm determines complete assignments that can be pruned from consideration when earlier time steps are examined. While the use of the algorithm at time step H is identical to standard DCOPs, the calculation of bounds in earlier steps is not trivial, as we specify below.[4]

(METHOD 3) CROSS TIME STEP BOUND UPDATES. The use of the SBB algorithm to solve subproblems allows reasoning across time steps by calculating an initial upper bound to each subproblem at time steps $1 \leq t < H$ from the results of its respective subproblem at time step H, thereby allowing immediate pruning of dominated partial assignments.

(METHOD 4) CROSS TIME STEP SINGLETON NOGOODS. In this method, the algorithm learns about *singleton nogoods* to each subproblem at time steps $1 \leq t < H$ from its respective subproblem at time step H. Singleton nogoods are values of the domain of an agent that are assigned to its variable only in dominated solutions. Removing such singleton nogoods from the domains of variables prevents redundant examination of dominated solutions. This method is inspired by the unconditional deletion method proposed in [3]. Yet, to best of our knowledge this method has not been previously used for cross time step reasoning in dynamic problems.

7 Experimental Evaluation

In order to evaluate the improvement of the proposed enhancement methods over the naive algorithm, we implemented the naive version of the algorithm, added each of the methods, and measured the effect. In all of our experiments, each agent owns one variable with domain size 3. We randomly select the constraint costs from [1, 1000], set the horizon to 3, and set the number of dynamic combinations for each agent to 3 chosen from the following possible dynamic elements: Between subproblems in different time steps, (1) agents can appear or disappear; (2) the values in the domains of variables can change; and (3) the constraints costs can change.

Our first set of experiments compared the performance of the different versions of the algorithm. The results reported for this experiment are an average of the algorithms' performance solving the same 70 random R_DCOPs – Once with a maximum number of 6 agents and once with a maximum number of 5 agents. The cost of change (for both γ_{-} and γ_0 separately) was randomly selected from [1,100]. For a better understanding of the complexity of the R_DCOPs being solved, note, for example, that a single R_DCOP with a maximum number of 6 agents is equivalent to solving 1,594,324 static DCOPs in the worst case,[5] without accounting for cross time step checks.

Figure 3 presents the runtime of the algorithm with the different pruning methods enhancing it. The results include only the average results of experiments in which the algorithm solve the problems within 30 minutes at *all* pruning levels (including the naive version). Since these results do not include the cases where the naive version

[4] While there are more advanced DCOP algorithms, we choose SBB because it easily allows the exploitation of bounds that were inferred in different time steps, which is a delicate task.

[5] The maximum number of variables in P_0 is 6. Therefore, there are 3^6 solutions for P_0 that need to be examined. For each one, we need to examine $H \times |\Delta| = 3\times$ subproblems across all time steps greater than 0, in addition to solving P_0 at time step 0. To conclude we examine: $1 + (3^6) \times 3 \times 3^6 = 1,594,324$ static DCOPs.

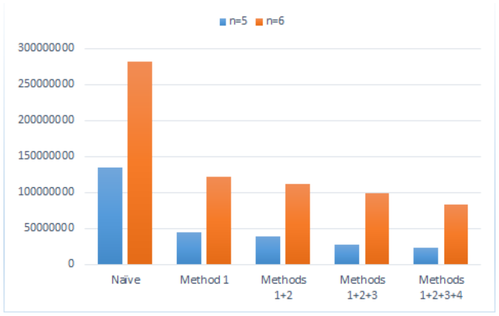

Fig. 3: Average Number of Non Concurrent Checks

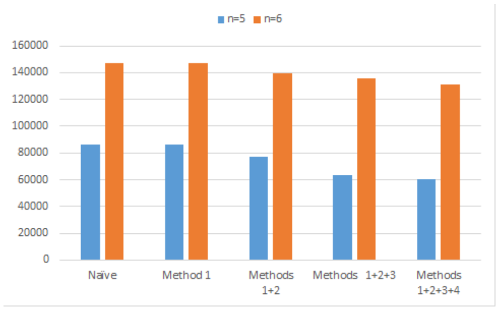

Fig. 4: Average Number of Messages

did not solve the problem but the version with some of the pruning methods did, the improvement presented is a *lower bound* on the actual improvement that the pruning methods offer. The number of checks reported for the algorithms includes both standard *Non-Concurrent Constraint Checks* (NCCCs) [13] that are performed when solving a subproblem and *Cross Time Step Checks* (CTSCs), which includes checks between problems of different time steps. The results presented are of experiments that include problems with 5 agents ($n = 5$) and with 6 agents. In each problem (as mentioned above), the number of possible dynamic combinations of events for each agent is 3. Thus, for 5-agent problems, $3^5 = 243$ subproblems must be solved at each time step, while in 6-agent problems, $3^6 = 729$ subproblems must be solved at each time step. Empirically, the combination of all the methods we propose for enhancing the naive algorithm reduces the number of constraint checks by 83% for $n = 5$ and by 70% for $n = 6$.

Figure 4 presents the total number of messages sent by agents in each version of the algorithm. Method 1 does not add any messages. Therefore, the number of messages it sends is identical to the number of messages sent by the naive algorithm. When adding each enhancement method, additional information is exchanged by the agents and the number of messages in each iteration grows, yet, the total number of messages is smaller because of the pruning of the search space that allows it to finish earlier (in a smaller number of iterations).

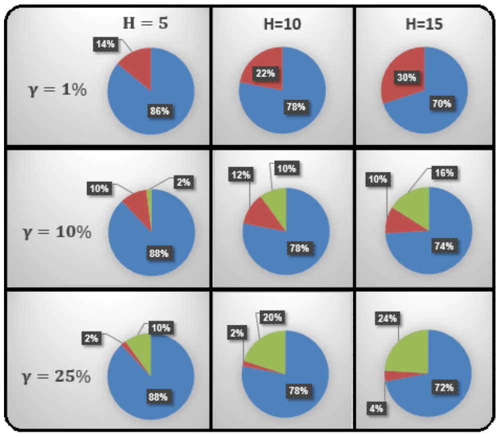

Fig. 5: Sensitivity Analysis Solved R_DCOPs

Next, we performed a sensitivity analysis for some of the parameters of the problem. This experiment includes all R_DCOPs with 5 agents. All parameters were set as in the first set of experiments unless specifically mentioned otherwise. Figure 5 presents the percentage of solved R_DCOPs out of 50 instances for different horizons $H \in \{5, 10, 15\}$ and for the maximal percentage of the maximal constraint cost for changing an assignment $\gamma \in \{1\%, 10\%, 25\%\}$. In red is the percentage of R_DCOPs that were solved only when using the algorithm with all proposed enhancement methods. In blue is the percentage of R_DCOPs that were solved by the naive algorithm as well. In green is the percentage of R_DCOPs that were not solved by any version within 30 minutes. The results demonstrate that the enhancement methods proposed are most effective when the horizon is larger and for smaller costs for replacing assignments. The reason is because the larger horizon allows the cross time steps bounds that we find to be used in more time steps. A smaller cost for change allows the our bounding methods to compute tighter bounds and prune more.

8 Conclusions

Many multi-agent scenarios are dynamic and require agents to make decisions in a decentralized way while taking into account the quality of their solution with respect to possible dynamic events. Dynamic DCOPs is one popular way for modeling such dynamic distributed scenarios. Existing work on this area assume that the agents can change their decisions each time the problem changes. In contrast, in this paper, we propose the Resilient DCOP (R_DCOP) model, where the objective is to find a single solution that is resilient towards future changes. We also proposed the first (naive) complete algorithm to solve R_DCOPs as well as enhancements to improve its performance. Our empirical results demonstrate that the combination of the proposed methods reduces the runtime of the algorithm by at least 70% and improves with increasing horizon. These contributions thus serve as important foundational first steps toward the modeling and deployment of Dynamic DCOP algorithms in applications where agents must make decisions that are *long-term commitments* independent of how the problem changes.

References

1. R. Becker, S. Zilberstein, V. Lesser, and C. Goldman. Solving transition independent decentralized Markov decision processes. *Journal of Artificial Intelligence Research*, 22:423–455, 2004.
2. D. Bernstein, R. Givan, N. Immerman, and S. Zilberstein. The complexity of decentralized control of Markov decision processes. *Mathematics of Operations Research*, 27(4):819–840, 2002.
3. C. Bessiere, P. Gutierrez, and P. Meseguer. Including soft global constraints in DCOPs. In *Proceedings of the International Conference on Principles and Practice of Constraint Programming (CP)*, pages 175–190, 2012.
4. J. S. Dibangoye, C. Amato, O. Buffet, and F. Charpillet. Optimally solving dec-pomdps as continuous-state mdps. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 2013.
5. J. S. Dibangoye, C. Amato, and A. Doniec. Scaling up decentralized mdps through heuristic search. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 217–226, 2012.
6. J. S. Dibangoye, C. Amato, A. Doniec, and F. Charpillet. Producing efficient error-bounded solutions for transition independent decentralized mdps. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 539–546, 2013.
7. H. Fargier, J. Lang, and T. Schiex. Mixed constraint satisfaction: A framework for decision problems under incomplete knowledge. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence and Eighth Innovative Applications of Artificial Intelligence Conference, AAAI 96, IAAI 96, Portland, Oregon, August 4-8, 1996, Volume 1.*, pages 175–180, 1996.
8. K. D. Hoang, F. Fioretto, P. Hou, M. Yokoo, W. Yeoh, and R. Zivan. Proactive dynamic distributed constraint optimization. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 597–605, 2016.
9. K. D. Hoang, P. Hou, F. Fioretto, W. Yeoh, R. Zivan, and M. Yokoo. Infinite-horizon proactive dynamic DCOPs. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2017.
10. L. Kaelbling, M. Littman, and A. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101:99–134, 1998.
11. R. Lass, E. Sultanik, and W. Regli. Dynamic distributed constraint reasoning. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, pages 1466–1469, 2008.
12. R. Mailler. Comparing two approaches to dynamic, distributed constraint satisfaction. In *Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 1049–1056, 2005.
13. A. Meisels, E. Kaplansky, I. Razgon, and R. Zivan. Comparing performance of distributed constraints processing algorithms. In *Proceedings of the Distributed Constraint Reasoning Workshop*, pages 86–93, 2002.
14. P. J. Modi, W. Shen, M. Tambe, and M. Yokoo. Adopt: asynchronous distributed constraints optimizationwith quality guarantees. *Artificial Intelligence*, 2005.
15. R. Nair, P. Varakantham, M. Tambe, and M. Yokoo. Networked distributed pomdps: A synthesis of distributed constraint optimization and pomdps. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, pages 133–139, 2005.

16. F. A. Oliehoek, M. T. J. Spaan, C. Amato, and S. Whiteson. Incremental clustering and expansion for faster optimal planning in dec-pomdps. *Journal of Artificial Intelligence Research*, 46:449–509, 2013.
17. A. Petcu and B. Faltings. Superstabilizing, fault-containing multiagent combinatorial optimization. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pages 449–454, 2005.
18. A. Petcu and B. Faltings. Optimal solution stability in dynamic, distributed constraint optimization. In *Proceedings of the International Conference on Intelligent Agent Technology (IAT)*, pages 321–327, 2007.
19. S. Ross, J. Pineau, S. Paquet, and B. Chaib-draa. Online planning algorithms for pomdps. *J. Artif. Intell. Res. (JAIR)*, 32:663–704, 2008.
20. E. Sultanik, R. Lass, and W. Regli. Dynamic configuration of agent organizations. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 305–311, 2009.
21. A. Tarim, S. Manandhar, and T. Walsh. Stochastic constraint programming: A scenario-based approach. *Constraints*, 11(1):53–80, 2006.
22. T. Walsh. Stochastic constraint programming. In *Proceedings of the 15th Eureopean Conference on Artificial Intelligence, ECAI'2002, Lyon, France, July 2002*, pages 111–115, 2002.
23. W. Yeoh, P. Varakantham, X. Sun, and S. Koenig. Incremental DCOP search algorithms for solving dynamic DCOPs (Extended Abstract). In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 1069–1070, 2011.
24. R. Zivan, H. Yedidsion, S. Okamoto, R. Glinton, and K. P. Sycara. Distributed constraint optimization for teams of mobile sensing agents. *Autonomous Agents and Multi-Agent Systems*, 29(3):495–536, 2015.