Decentralised Control of Intelligent Devices: A Healthcare Facility Study

Sacha Lhopital[1], Samir Aknine[2], Vincent Thavonekham[1], Huan Vu[2][0000−0002−0785−6907], and Sarvapali Ramchurn[3][0000−0001−9686−4302]

[1] VISEO Technologies, Lyon, France
{sacha.lhopital,vincent.thavonekham}@viseo.com
[2] Université de Lyon, CNRS, Université Lyon 1, LIRIS, UMR5205, Lyon 69622, France
samir.aknine@univ-lyon1.fr, huan.vu@liris.cnrs.fr
[3] University of Southampton, Southampton, United Kingdom
sdr1@soton.ac.uk

Abstract. We present a novel approach to the management of notifications from devices in a healthcare setting. We employ a distributed constraint optimisation (DCOP) approach to the delivery of notification for healthcare assistants that aims to preserve the privacy of patients while reducing the intrusiveness of such notifications. Our approach reduces the workload of the assistants and improves patient safety by automating task allocation while ensuring high priority needs are addressed in a timely manner. We propose and evaluate several DCOP models both in simulation and in real-world deployments. Our models are shown to be efficient both in terms of computation and communication costs.

1 Introduction

The penetration of novel Internet-of-things technology in the healthcare setting is growing rapidly. Many of these devices serve to monitor patients and alert healthcare professionals whenever abnormalities are detected (for instance, a syringe pump is going to ring when it detects an air bubble in its mechanism) or when routine checks are needed (about every four hours).

Nevertheless, operating and monitoring these devices take a considerable amount of time ranging from 5 to 10% a day. As a result, a healthcare provider has to check all those devices on a regular basis. As part of this project, a series of interviews with hospital staff were conducted. The following conclusions were drawn from these interviews. Every time a device encounters a technical problem (e.g. running out of power) or the device is ending its program (e.g. the syringe pump will finish its program in less than ten minutes), the device produces a very loud tone in order to alert the medical staff. Attending to such notifications rapidly becomes intractable with large departments with hundreds of patients. Thus, when a device rings, the medical staff cannot remember which one it is and what action is expected. Therefore, the staff needs to: (1) go in the room (2) notice the defective equipment (3) act accordingly (for instance: get a medication stored in another room). Limiting these tasks (by only checking rooms requiring an intervention, where it used to be every room of the department every x hours for instance) can improve healthcare professionals' work conditions. With this

in mind, several issues were noticed: (1) most of the devices are not integrated into an information system, thus forcing healthcare professionals to individually check them regularly, thereby wasting precious time on monitoring actions; (2) audible notifications from multiple devices at the same time can raise levels of stress and confusion among staff.

Against this background, in this paper we propose a novel approach that looks to minimise the intrusiveness of such devices. Specifically, we develop a solution that: (1) Detects anomalies and manages tasks division by combining data from multiple sources; (2) Constructs and suggests an action plan for healthcare provider. The aim is to provide staff with situational awareness and help them anticipate future interventions. The purpose of our system is to warn the medical staff before devices ring, but without increasing the frequency of their interventions. We formulate the problem as a Distributed Constraint Optimization Problem (DCOP) which has been shown to be effective in itinerary optimization [6], [15] and scheduling problems [7]. This decentralized approach has the benefit of distributing the main computations across all available devices. Specifically, this paper advances the state of the art in the following ways.

1. We propose a DCOP approach that limits the amount of data transmitted to a central node.
2. Our DCOP approach allows the seamless integration or removal of IoT devices.
3. We show that a DCOP approach is a natural way of modelling the problem. Our system was simulated using Raspberry Pi's to help represent the problem in a more realistic way. This specific deployment method is very important since it adds development constraints to our system (execution time, machine resources, interactions with the staff).

The remainder of this article is organized as follows. First, the paper introduces the problem statement. After that, we describe the DCOP model we propose to solve this problem. Then, we detail the proposed solution. We pursue with an evaluation of our work. In the conclusion, we summarize the work done and we provide some perspectives.

2 Related Work

MobiCare [1], designed by Chakravorty in 2006, provides a wide-area mobile patient monitoring system to facilitate continuous monitoring of the patients physiological status. Like CodeBlue [9], [11] is a popular healthcare project based on BSN framework (Body Sensor Network). In this system, sensors on the patient's body transmit information wirelessly to other devices for further analysis (like laptops and personal computers). While CodeBlue is using a wireless architecture, there have been many efforts in the medical field to design gateways for specific applications. For example, [2], [16] suggest the use of gateways instead of wireless or Ethernet to connect networks with different protocols.

Beyond these first solutions, DCOP algorithms have already been tested in practical scenarios such as travel optimization ([6], [15]) or planning ([7]). Among the DCOP algorithms, three are particularly well known: ADOPT [12], DPOP [14], OptAPO [10].

[6] compared these main algorithms in situations where the environment changes dynamically. Their study shows that these algorithms offer good performance but also highlight some limitations. DPOP is the fastest algorithm at runtime, but it is extremely greedy about the size of the messages exchanged. ADOPT gives variable results depending on the constraints. Indeed, if the system is not subject to many conflicting constraints, the algorithm will be efficient. On the other hand, if many constraints conflict, ADOPT does not provide efficient results. OptAPO was proven to be incomplete and therefore, a complete variant has been proposed [5]. Both variants are based on a mediator agent, so the resolution is not fully distributed. [7] proposed another algorithm to solve dynamic problems called DCDCOP (*Dynamic Complex DCOP*) based on a case study of time use optimization in a medical context. This algorithm - mainly based on the addition of a *Degree of Unsatisfaction* measure - dynamically guides agents through the resolution process. This method is more appropriate where agents try to optimize several variables at the same time.

The mechanism we propose is based on the DPOP algorithm [13]. We propose an improved version of the algorithm proposed by [13]. We have designed new heuristics for the *Depth First Search* (DFS) tree generation to improve the execution speed. This is the first model for device management using DCOPs.

3 Problem Statement

We consider a hospital facility made up of several departments. Each department includes a set of rooms. Within a room, several devices are deployed to monitor the status of each patient. Each room has a neighborhood formed by a set of rooms. The objective of the system is to determine the times when healthcare providers pass through each of these rooms and prioritize them in order to perform the operations required for each patient (e.g. recharging a syringe pump, etc.). The intervention time consists of a number of minutes before the situation becomes critical. Thus, we prioritize the rooms depending on which one is the most urgent. The emergency "level" is calculated dynamically and depends on several parameters. A room deadline is the time when a device of this room will ring. We define, a configuration O_t as the set of times for all the rooms in the department for a time step t. Each configuration O_t must satisfy the following rules: (1) a room must have only one intervention time at a time t_i; (2) the current configuration must be accessible by all rooms so that they share the same information; (3) all the rooms should not call the healthcare providers at the same time, except the rooms in the same neighbourhood. In order to build this configuration, we model the problem as follows. Let $A = \{a_1, ..., a_n\}$ be the set of agents (which manage the rooms) with n the total number of rooms. Each room i is modeled by an agent a_i. For a room i, each agent a_i will compute a value v_i which represents the time (in minutes) before the next intervention. Let $D = \{d_1, ..., d_n\}$ be all the possible values for v_i. If $d_i = +\infty$, then no intervention is required in the room i. Let t be the current time step. A configuration $O_t = \{v_1, ..., v_n\}$ is optimal if, and only if, it respects all the constraints of each room in the department. The system's inputs are the agents A and the previous configuration O_{t-1}. Thus, the O_t configuration is optimal if F, the interventions cost function, is

minimal according to the next intervention dates v_i. Our goal is to seek a minimization:

$$\arg\min_{O_t} F(O_t) = \sum_{i=1}^{n} C_i \tag{1}$$

For a specific v_i, C_i returns a global cost $\in \mathbb{R} \cup \{\infty\}$. This global cost, to satisfy F, is defined regarding all the following structural constraints.

- **c1. Device number constraint:** If there are no devices in the room, the agent will not ask for intervention: Let $M_i = \{m_1, ..., m_l\}$ be the set of all the devices of the room i.

$$\forall a_i \in A, |M_i| = 0 \Rightarrow v_i = +\infty \tag{2}$$

- **c2.a Simple device rescheduling constraint:** Healthcare professionals have to check the room just before a device ends its program. We define the function $isInCriticState(m_i)$ which returns $true$ if the device m_i is in a critical state.

$$\begin{aligned} &\forall a_i \in A, \forall m_l \in M_i, \\ &isInCriticState(m_l) \Rightarrow v_i < 10 \end{aligned} \tag{3}$$

- **c2.b Critical device rescheduling constraint:** Healthcare professionals have to reschedule machines when they come to a critical state. Therefore, we define the function $programState(m_i)$ to return the remaining time (minutes) before m_i ends its program. If the remaining time is not computable (i.e. the device does not require an intervention), the function returns $+\infty$. As a consequence, if a device in the room i ends its program in less than 30 minutes than v_i has to be less than this value.

$$\begin{aligned} &\forall a_i \in A, \exists m_l \in M_i, \\ &\neg\, isInCriticState(m_l) \wedge programState(m_l) \leq 30 \\ &\Rightarrow v_i \leq programState(m_l) \end{aligned} \tag{4}$$

- **c3. Neighbourhood constraint:** If two rooms are in the same location, they can synchronize their decisions to avoid multiple interventions in a short time. In other words: two neighbours should not ask for interventions with less than $t_{synchro}$ minutes interval (except if they both call at the same time). We define the function $neighbours(a_i, a_j)$ as a function returning $true$ when the agents a_i and a_j are neighbours.

$$\begin{aligned} &\forall a_i, a_j \in A^2, neighbours(a_i, a_j) \Rightarrow \\ &(|v_i - v_j| > t_{synchro} \vee |v_i - v_j| = 0) \end{aligned} \tag{5}$$

- **c4.a Patient's condition constraint:** If a patient uses multiple devices (more than five), we consider that he needs more attention than others. His state should be checked at least every three hours instead of four otherwise. Let τ_i be the elapsed time since a medical staff came into the room i.

$$\forall a_i \in A, (|M_i| > 5 \wedge \tau_i \geq 180) \Rightarrow v_i < 30 \tag{6}$$

– **c4.b Time between two visits constraint:** The elapsed time between two interventions in a room cannot exceed four hours. This constraint is derived from the interviews we made. If more than 3 hours and 30 minutes have passed since the last visit, the system should plan another visit within 30 minutes.

$$\forall a_i \in A, (|M_i| \geq 1 \wedge \tau_i \geq 210) \Rightarrow v_i < 30 \tag{7}$$

– **c5. Quietness constraint:** Each agent a_i verifies if there is no device in the room i that needs intervention. If it is the case, healthcare staff can ignore the room:

$$\begin{aligned} &\forall a_i \in A, \forall m_l \in M_i, \\ &(\neg isInCriticState(m_l) \wedge \\ &programState(m_l) > 30 \wedge \tau_i < 180) \\ &\Rightarrow v_i \geq 240 \end{aligned} \tag{8}$$

Example 1 *Consider the following scenario with 6 rooms (a_1, a_2, a_3, a_4, a_5 and a_6) in a medical department. None of the rooms has devices to monitor except the room a_3 which is monitoring 3 devices programmed to end respectively in 40 minutes, 60 minutes, and the last is in a critical state. The neighbourhood is the following: a_1 is surrounded by a_3 and a_2 ; a_4 is surrounded by a_3 and a_5 ; And a_5 is also neighbour with a_6. Therefore, a_1, a_3, a_4, a_5 have two neighbours, while a_2 and a_6 have only one neighbour (cf. Figure 1).*

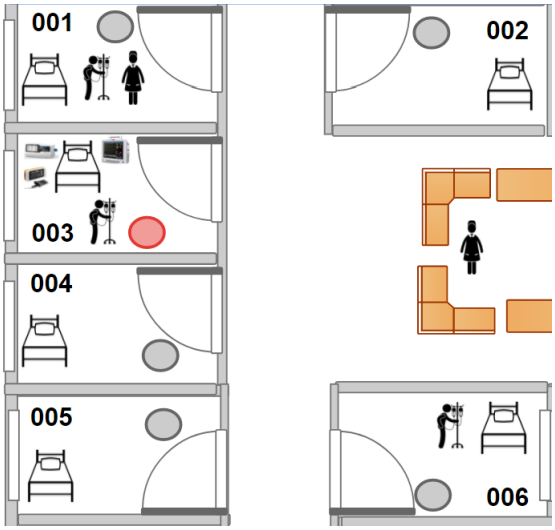


Fig. 1: Illustration scenario with the 6 rooms in the medical department. None of the rooms have devices to monitor except agent a_3 which is monitoring 3 devices programmed to end respectively in 40 minutes, 60 minutes, and the last (in red) is in a critical state. All adjacent rooms are neighbours. Therefore, a_1, a_3, a_4, a_5 have two neighbours, while a_2 and a_6 have only one neighbour.

Also, assuming that:

- $M_3 = \{m_{31}, m_{32}, m_{33}\}$
  with $isInCriticState(m_{32}) = True$ *(the device is in a critical state),* $programState(m_{31}) = 60$ *and*
  $programState(m_{33}) = 40$.
- $M_4 = M_5 = M_1 = M_2 = M_6 = \{\emptyset\}$.
- $\tau_3 = 60$.
- *We also set* $t_{synchro} = 30$.

The structural constraints can be described as:

c1: v_4, v_5, v_1, v_2 *and* v_6 *will take the value* $+\infty$.
c2: *The device* m_{32} *needs intervention. Thus,* $v_3 < 10$. *The device* m_{31} *ends its program in 60 minutes and the device* m_{33} *in 40 minutes. Hence:* $v_3 \leq 60$*;* $v_3 \leq 40$.
c3: *Given the neighbourhood in the scenario, we deduce that* $|v_3 - v_4| > t_{synchro}$ *and* $|v_3 - v_1| > t_{synchro}$.
c4: *The last intervention is very recent (because* $\tau_3 < 180$ *and* $|M_3| < 5$*), so this constraint will not be applied.*
c5: *This constraint does not apply here.*

Next, we formalize the problem as a distributed constraint optimization problem.

Centralized solutions to scheduling have a lack of scalability and adaptability to dynamic events such as the arrival of an emergency. In such a dynamic context, using a decentralized approach allows to be proactive to any change of the devices.

4 DCOPs for Device Management

We formalise the Device Management DCOP as a tuple $\{A, V, D, C\}$, where: $A = \{a_1, a_2, ..., a_n\}$ is a set of n agents; $V = \{v_1, v_2, ..., v_n\}$ are variables owned by the agents, where variable v_i is owned by agent a_i; $D = \{d_1, d_2, ..., d_n\}$ is a set of finite-discrete domains. A variable v_i takes values in $d_{v_i} = v_1, ..., v_k$; $C = \{c_1, ..., c_m\}$ is a set of constraints, where each c_i defines a cost $\in \mathbb{N} \cup \{\infty\}$. A solution to the DCOP is an assignment to all variables that minimizes $\sum_{i=1}^{n} c_i$.

DCOP is a preferred solution to deal with stochastic and dynamic environments with data gathered from different agents. It is applied to numerous different applications in multi-agent systems such as disaster management, meeting scheduling, sensor network [3].

There are several ways to formalize our problem as a DCOP, depending on what agents, variables and constraints are representing. Here we present three approaches to formalize the medical optimization problem as a DCOP: a fully decentralized room-based approach (**Room Approach**), a semi-decentralized area-based approach (**Area Approach**) and finally a semi-decentralized multi-variable approach (**Multi-variable Area-based Approach**). We intend to show the effect of different levels of decentralization on the quality of time computing. We evaluate and show the performance of each approach which may be suitable for different medical device conditions.

4.1 Room-based Approach

The **Room Approach** consists of modelling all the rooms as agents. The number of agents corresponds to the number of rooms to monitor. Each agent has a variable that corresponds to the desired intervention time, depending on the devices conditions in the room. The domain of the variables varies from 0, which is the most critical call, to ∞ which means that no call is planned. We then map the structural constraints described in the equations 2 to 8 as follows:

Device number constraint

$$c_1(v_i) = \begin{cases} \infty, & \text{if } |M_i| = 0 \text{ and } v_i \neq +\infty \\ 0, & \text{otherwise} \end{cases} \tag{9}$$

Device rescheduling constraint

$$c_2(v_i) = \begin{cases} \infty, & \text{if } \exists m_l \in M_i, \\ & isInCriticState(m_l) \text{ and } v_i \geq 10 \\ 1, & \text{if } \exists m_l \in M_i, \ (\neg isInCriticState(m_l) \\ & \text{and } programState(m_l) \leq 30) \\ & \text{and } v_i > programState(m_l) \\ 0, & \text{otherwise} \end{cases} \tag{10}$$

Neighbourhood constraint

$$c_3(v_i, v_j) = \begin{cases} 1, & \text{if } neighbours(a_i, a_j) \\ & \text{and } |v_i - v_j| \leq t_{synchro} \\ & \text{and } |v_i - v_j| \neq 0 \\ 0, & \text{otherwise} \end{cases} \tag{11}$$

Patient's condition and time between two visits constraint

$$c_4(v_i) = \begin{cases} \infty, & \text{if } ((|M_i| > 5 \text{ and } \tau_i \geq 180) \\ & \text{or } (|M_i| \geq 1 \text{ and } \tau_i \geq 210)) \text{ and } v_i \geq 30 \\ 0, & \text{otherwise} \end{cases} \tag{12}$$

Quietness constraint

$$c_5(v_i) = \begin{cases} 1, & \text{if } \forall m_l \in M_i, \ v_i < 240 \\ & \text{and } \neg isInCriticState(m_l) \\ & \text{and } programState(m_l) > 30 \text{ and } \tau_i < 180 \\ 0, & \text{otherwise} \end{cases} \tag{13}$$

The objective of our DCOP is to minimize $\sum_{i=1}^{n} c_i$. This optimization represents the goal of the system (i.e. minimizing the number of rooms making calls without violating any structural constraint).

4.2 Area-based Approach

Instead of considering each room as an agent, we can consider a hospital area as an agent which monitors multiple rooms. We consider that the department is divided into several areas. As an area agent, it holds a unique variable v that contains the most critical intervention time among all monitored rooms. By gathering information from several rooms in this way, the system solves the problem using the same constraints as in the previous approach. However, these constraints are applied to all the rooms instead of a single room. On the other hand, the neighbourhood constraint no longer concerns the rooms, but rather the areas. Therefore, the global cost C_i is also computed by area as follows:

$$C_i = \begin{cases} \infty, & \text{if } \exists r_k \in R_i,\ \exists m_l \in M_k \\ & isInCriticState(m_l) \\ \sum_{k=1}^{p} \sum_{q=2}^{m} c_q(v_i), & \text{otherwise} \end{cases} \tag{14}$$

In this approach, the device number constraint is no more used. We define a similar constraint to take all rooms R_i into account and no longer a single one.

This approach also requires us to consider the impact on privacy and, more specifically, the transit of data. For security and data protection reasons, the centralization of data is very sensitive.

4.3 Multi-variable Area-based Approach

To go further in the area-based modelling, a slightly more specific approach was also considered where each area defines an intervention time for each room. In this last approach, we still consider that each one of the rooms in the area has the knowledge on all the other rooms. Thus, we still have a single area agent, but this agent will calculate a time set $V_{a_i} = \{v_{r_1}, v_{r_2}, ..., v_{r_k}\}$ where k is the number of rooms of the area i. This approach also requires us to consider the impact on privacy, like the area-based approach.

Now that we have formalized the problem as a DCOP, we discuss the quality of the solution and the usefulness of the method.

5 A DPOP Solution for the Device Management Problem

To solve the DCOP presented above, we use the DPOP algorithm (*Dynamic Parameter Optimization Problem*) [14], based on the exchange of messages between agents. We chose to use DPOP as it is one of the fastest DCOP algorithms [6], working by *tree aggregation* [4]. In more details, DPOP operates on a matrix handling algorithm. To communicate agents use a tree graph (DFS): an undirected graph, which contains a variable node v_i for each agent, and an edge connecting a variable node v_i with another variable node v_j if and only if v_i is a neighbour of v_j. Each agent in DPOP takes the role of the variable node which represents its own variable. Figure 2 shows the tree graph of the room-based approach for the scenario presented in the Example 1. The main process of DPOP consists in computing and exchanging messages between variable nodes through the tree graph constructed. At each iteration k of the process,

all agents execute 3 phases. In the first phase, a proper tree graph is generated to serve as a communication structure for the next steps. To do so, agents exchange messages through their neighbourhood in order to generate the tree graph. When this graph is complete, the second phase starts: starting from the leaves of the tree, each agent a_i computes its own cost matrix $Util_i$ (depending on its v_i value and on its children v values) and propagates it upward through the tree edges. Those matrices summarize the influence of the sending agent and its neighbours on the next steps. The third phase is started by the root when phase 2 is over. Each agent computes its optimal value for v based on the $Util$ matrix and the $Value$ message it received from its parent. Then it sends this value as a $Value$ message to its own children.

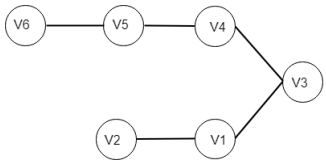


Fig. 2: Room-based approach tree graph for the scenario presented in Example 1. There are 6 agents (v_1 to v_6), each is connected to its neighbours.

Example 2 *Consider the tree graph presented in Figure 2. Let $D = \{0, 30, 241\}$. The message that the variable v_2 sends to its parent v_1 for the iteration k is the following: $Util_2 = \begin{bmatrix} a \\ b \\ c \end{bmatrix}$, where each matrix value is the result of a cost ($\sum_{l=1}^{5} c_l$). The abscissa axis represents the lowest possible value for v_2, while the ordinate represents all possible values for v_1 in D. When v_1 is set up with the best value for this iteration (let us say b), v_1 sends this value back to v_2.*

During the propagation of messages, an agent is able to calculate locally its next intervention time that minimizes the sum of the costs over all neighbours functions. Classic DPOP does not always guarantee convergence at the end of an iteration. In our context, we overcome this issue with the use of the **Quietness constraint**.

Event Detection and Management Dynamic events should be taken into account during the resolution. For instance, we need to detect when a medical staff is in a room (or when she is near). Every event will dynamically impact some constraints. Thus, the system will detect healthcare staff interventions and can update the different constraints parameters, for instance, the time since the last intervention (τ), the states of the devices, the number of interventions ($|M|$, $isInCriticState(m)$, $programState(m)$).

Event detection is essential when a device enters in a critical state. The healthcare provider needs to be called right away because the situation corresponds to an emergency.

Priority Management After each iteration, all agents set their next intervention time v depending on their knowledge about their devices. However, if an agent i asks for a quicker intervention than its neighbours, i will not necessarily be satisfied if it is not the most critical in the system.

In order to deal with this issue, we define the concept of priority for the DCOP solver. The classic DPOP algorithm generates a DFS tree for the problem to solve. But for the same problem, several DFS trees may be generated. Yet, depending on the generated tree, the algorithm finds a local solution (possibly the best solution, but not necessarily).

Yet, in our case study, finding a local solution is not enough. We therefore search for the best solution because healthcare providers need to check on the most urgent patients first. To do so, we define some specific rules for the DPOP that allow agents to declare themselves as more important. More precisely, those rules impact the tree graph construction in the first phase of DPOP by putting the most important agents at the top of the DSF tree. This allows them to choose their intervention time first. Three specific priority rules are defined:

- **Critical Priority** is triggered when a device enters a critical state. The concerned agent will ask other agents to start a new DCOP computation handling its condition. This priority is the most important. When this rule applies, it overcomes all the others. When triggered, all agents will start a new computation of the DCOP algorithm.
- **Time Priority** is triggered when a device needs intervention since the last iteration, but no healthcare provider has been able to intervene. At each iteration, the agent will increase its priority until a healthcare provider answers the call.
- **Intervention Consequence Priority** is triggered after a healthcare assistant provides an intervention. When triggered, the priority of the concerned agent is reduced to the lowest value.

6 Empirical Evaluation

We have evaluated the performance of our method using the DPOP algorithm. The algorithm was implemented in Python 3.6 and deployed on Raspberry Pi devices, using Broadcom BCM2837 64-bit processor with four ARM Cortex-A53 hearts - 1,2 GHz. The use of Raspberry pi allows us to physically distribute our agents - as it will be the case in a real situation. DPOP was also implemented using Frodo [8]. In order to communicate, the Mqtt protocol was used with a Mqtt Server running on a local gateway. All compared values are averages on up to 10 to 50 consecutive simulations (the exact number depends on the used method with the Frodo simulator or with the deployed system and their multiple parameters). All algorithms are evaluated according to their execution time. We ran our experiments with all our different approaches: room-based approach, area-based approach and the multi-variable area-based approach.

6.1 Benchmarking

The Figure 3a represents the execution time of each approach: **Room Approach** in solid line; **Area Approach** in big dots; and **Multi-variable Area-based Approach**

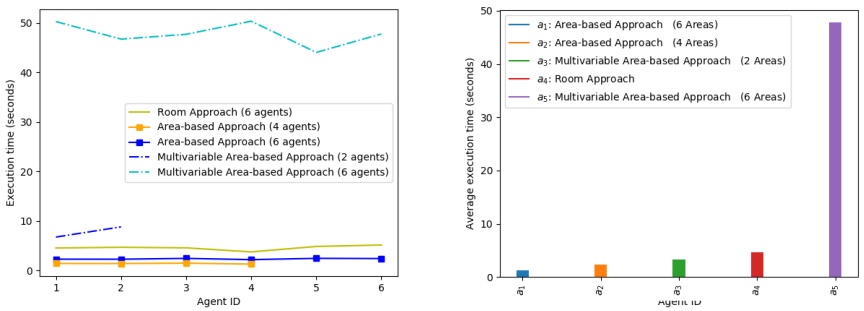
Algorithm 1 Scheduling Agent pseudo-code

```
Require: root = NULL
  for all agents do
    if root == NULL then
      send starting signal
    else
      send starting signal with root as a parameter
    end if
  end for
  while results size < number of agents do
    wait
  end while
  for all results do
    if result == 0 then
      root = agent
      Start over
    end if
    if result < 30 and agent.priority > 0 then
      agent.priority = agent.priority + 1
    end if
    if result > 30 and agent.priority > 0 then
      agent.priority = 0
    end if
  end for
```

Algorithm 2 DPOP Agent pseudo-code when receiving a message to start

```
  if root != NULL then
    run DPOP algorithm with root as the DFS Tree root
  else
    run DPOP classic algorithm
  end if
  return result
```

with dashed lines. Multiple curves are shown for different numbers of agents in the system. Whatever the situation, these curves show that the fastest approach is the **Area Approach** (execution time between 1,29 and 2,68 seconds by agent). The **Room Approach** also gives good results (between 3,75 and 5,6 seconds). This evaluation shows that the **Area Approach** can offer faster results but it will be at the detriment of the precision of the results. The Figure 3b summarizes these results. Figures 5a and 6a present the results with 6 agents. Among them, 3 agents run on Raspberry Pi devices and 3 agents run on Windows 10 computers. Figures 5b and 6b show a similar situation but with 10 agents (3 agents on Raspberry Pi devices and others on Windows 10 computers). Also, dotted lines represent a specific simulation performed with simulated Raspberry Pi devices (QEMU). The use of this simulator drastically increased the execution time because this simulator uses more computer resources. Therefore, we will focus on the analysis of the results represented in solid lines. The total execution time



(a) (b)

Fig. 3: Evaluations of our algorithm. (a) Execution time of the system deployed on Raspberry Pi devices depending on the approach for a medical department of 6 rooms. (b) Average execution time of the system deployed on Raspberry Pi devices depending on the approach for a medical department of 6 rooms.

of our deployed system is higher than the execution time we observed with the DPOP simulation (cf. Figure 4). For 6 agents (cf. Figure 5a), we provide a solution in less than 1 second using the simulator and in less than 5 seconds with our deployed system. This can be explained by the fact that our system provides an optimal solution, whereas FRODO provides a local optimum. This is the case because the "priority constraint" cannot be taken into account by Frodo without completely rewriting the DPOP algorithm. Secondly, regardless of the number of agents, the execution time is quite similar. Against the **Room Approach**, the **Area Approach** runs the algorithm much faster. For instance, 10 rooms divided into 4 areas give results in 1,64 seconds, and 10 rooms divided into 6 areas give results in 2,15 seconds. This semi-decentralized approach allows the system to produce more relevant results for great numbers of rooms (more than 30). Furthermore, the algorithm gives 4,64 seconds for 50 rooms divided into 4 areas and 9,29 seconds for 6 areas. This is consistent since the system only has 4 to 6 agents instead of 50 agents for the **Room Approach**.

Table 1 gives the execution times for the **Multi-variable Area-based Approach**. If the method takes more time to execute, we observed some considerable differences between two agents, execution times for the same iteration. For instance, in the table 1, agent 6 takes 1571 seconds to execute and agent 4 takes 129 seconds while others take less than 65 seconds. Those major differences can be explained by the new data structure that the agents use. Indeed, instead of using matrices of $|D|$ dimensions, the agents are computing matrices of $|D|^{NbRoomsArea(a_i)}$ dimensions where $NbRoomsArea(a_i)$ is the number of rooms managed by the agent a_i. Therefore, depending on their positions in the tree graph in the DPOP algorithm, some agents will have to deal with much more data because each parent in the tree will receive as much matrix $|D|^{NbRoomsArea(a_i)}$ as it has children. The resulting matrix will then have the

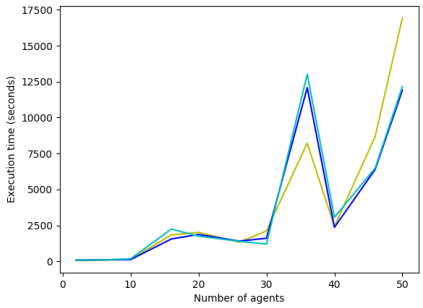


Fig. 4: Execution time depending on the number of agents using FRODO.

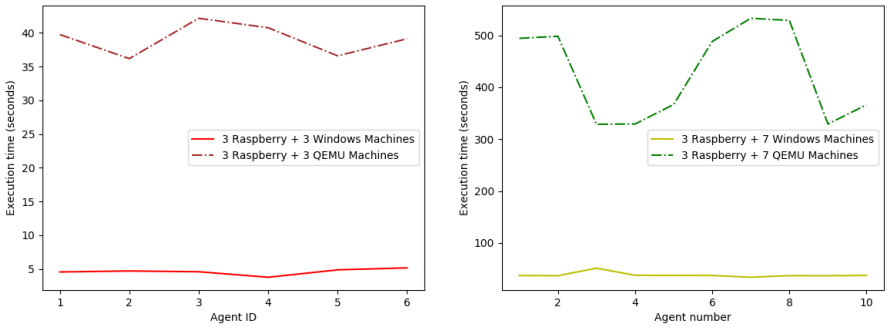
following dimension: $|D|^{NbRoomsArea(a_i)} \times NbChildren(a_i)$ where $NbChildren(a_i)$ is the number of children for the agent a_i in the DFS tree.

Area	1	2	3	4	5	6
Time (s)	66.95	66.67	91.12	9.36	0.99	1571.99

Table 1: Execution time for 12 Rooms and 6 Areas (with a **Multi-variable Area-based Approach**).

6.2 Message Size

Our algorithm (and DPOP in general) requires a linear number of messages. This is explained by the DFS construction which requires $2 \times l$ messages, where l is the number of edges in the tree graph. For n agents, *Util* DPOP phase requires $n - 1$ messages (from the bottom to the top of the tree). The *Value* propagation requires $n - 1$ messages (from the top to the bottom of the tree). The maximum message size and memory requirements grow exponentially with the number of agents. More precisely, DFS and *Value* messages are size and memory linear. But the complexity lies in the size of the *Util* messages, which is space and time exponential. Figures 6a and 6b give the size of the exchanged messages between the agents (respectively for 6 agents in the system, and for 10 agents). Those curves show the average size of the received messages for each agent depending on different tested situations. Our system exchanges very tiny messages compared to a Frodo simulation. For example with 10 agents (plain curves), the agent number 3 is the one receiving huge messages (average of 3800 bytes). But with Frodo, average messages size is in the order of 102 Kbytes. These results are explained by the fact that we use the Mqtt communication protocol, which allows to define a specific message structure. Also, regarding the Figures (6a and 6b), we observe



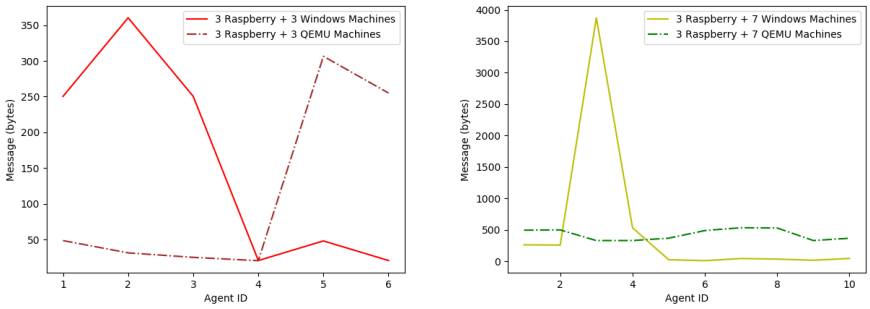
(a) (b)

Fig. 5: Execution time comparison between our system and QEMU simulations. (a) Comparison for 6 agents. The figure shows the quality of the solutions and that our system performed better on Raspberry Pi devices. (b) Comparison for 10 agents.

important variations from one agent to another. Those results are explained by the algorithm processing method. Indeed, depending on the DFS tree generated during the first phase, the agents at the top of this tree will receive bigger messages because their children will send them bigger matrices during the *Util* propagation phase. The *Util* matrix is a multidimensional matrix, with one dimension for each variable of the problem to solve. Therefore, every time an agent received a *Util* matrix from one of its children, the current agent increases the dimensions of the matrix (because the agent adds its own variable to solve to the matrix). In the DPOP algorithm as described by A. Petcu, each dimension of the matrix corresponds to the possible value of an agent. Each agent who receives a matrix from one of these children (in the DFS tree) actually receives a cost matrix based on the value the child will take. The more the matrix goes up in the DFS tree, the more children are to be taken into account, so the more dimensions there are. For instance, in the Figure 6b, agents 1 to 3 received bigger messages than other agents because they are at the top of the tree and they have to compute more data. The same effect is observed for the agent 2 in the Figure 6b.

7 Conclusion

In this paper, we have proposed an intelligent system to ease the daily work of the medical staff in helping patients. Our work offers a new method to supervise and monitor the various devices running in the rooms of the medical department and leaves the medical staff to focus on their patients. We provided a DCOP formalization of the problem and showed how we use the DPOP algorithm to solve it. Our method produces an efficient solution in terms of alerting healthcare professionals in intervention times. We showed the robustness of this solution to dynamic events. We also provided different formulations of the model with different degrees of privacy preservation when it comes to the



(a) (b)

Fig. 6: Evaluation of the messages size. (a) Average received messages size comparison between our system deployed on Raspberry Pi devices and our system deployed with QEMU simulator (for 6 agents). (b) Average received messages size comparison between our system deployed on Raspberry Pi devices and our system deployed with QEMU simulator (for 10 agents).

messages passed around. While our work has shown the potential of the DCOPs to solve the medical device management problem, in the future, we aim to extend our method to consider a more sophisticated system with cameras to detect healthcare providers movements in the medical department. More precisely, we want to deploy our system on Arduino instead of the Raspberry to follow up on this work. We also aim to test our system with real devices like syringe pumps and multi-parameter monitor as they are the most common ones in most medical services.

References

1. Chakravorty, R.: A programmable service architecture for mobile medical care. In: Proc. 4th Annu. IEEE Int. Conf. Pervasive Comput. Commun. Workshop (PERSOMW). pp. 531–536. IEEE Computer Society, Pisa, Italy (March 2006)
2. Emara, K., Abdeen, M., Hashem, M.: A gateway-based framework for transparent interconnection between wsn and ip network. In: Proceedings of the EUROCON. pp. 1775–1780 (2009)
3. Fioretto, F., Pontelli, E., Yeoh, W.: Distributed constraint optimization problems and applications: A survey. Journal of Artificial Intelligence Research **61**, 623–698 (2018)
4. Freuder, E.C.: A sufficient condition for backtrack-bounded search. Journal of the ACM (JACM) JACM Homepage archive **32**, 755–761 (October 1985)
5. Grinshpoun, T., Meisels, A.: Completeness and performance of the APO algorithm. Journal of Artificial Intelligence Research **33**, 223–258 (2008)
6. Junges, R., Bazzan, A.L.C.: Evaluating the performance of dcop algorithms in a real world, dynamic problem. In: Proc. of 7th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2008). pp. 599–606. Padgham, Parkes, Müller and Parsons (eds.), Estoril, Portugal (May 2008)

7. Khanna, S., Sattar, A., Hansen, D., Stantic, B.: An efficient algorithm for solving dynamic complex dcop problems. In: Proceedings - 2009 IEEE/WIC/ACM International Conference on Intelligent Agent Technology, IAT 2009 (October 2009)
8. Léauté, T., Ottens, B., Szymanek, R.: Frodo 2.0: An open-source framework for distributed constraint optimization. In: Proceedings of the IJCAI'09 Distributed Constraint Reasoning Workshop (DCR'09). pp. 160–164. Pasadena, California, USA (2009)
9. Lorincz, K., Malan, D.J., Fulford-Jones, T.R.F., Nawoj, A., Clavel, A., Shnayder, V., Mainland, G., Welsh, M., Moulton, S.: Sensor networks for emergency response: Challenges and opportunities. IEEE Pervasive Computing **3**, 16–23 (October – December 2004)
10. Mailler, R., Lesser, V.: Solving distributed constraint optimization problems using cooperative mediation. In: Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems. pp. 438–445. IEEE Computer Society, New York, USA (2004)
11. Malan, D., Fulford-Jones, T., Welsh, M., Moulton, S.: Codeblue: An ad hoc sensor network infrastructure for emergency medical care. In: Proc. MobiSys Workshop Appl. Mobile Embedded Syst. (WAMES). pp. 1–8. Boston, MA, USA (June 2004)
12. Modi, P.J., Shen, W., Tambe, M., Yokoo, M.: Adopt: Asynchronous distributed constraint optimization with quality guarantees. Artificial Intelligence **161**, 149–180 (January 2005)
13. Petcu, A.: Dpop, a dynamic programming optimization protocol for dcop. In: A Class of Algorithms for Distributed Constraint Optimization. pp. 52–57. IOS Press BV, Amsterdam, Netherlands (2009)
14. Petcu, A., Faltings, B.: A scalable method for multiagent constraint optimization. In: Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence. pp. 266–271. Professional Book Center Edinburgh, Scotland (2005)
15. Vu, H., Aknine, S., Ramchurn, S.D.: A decentralised approach to intersection traffic management. In: IJCAI, International Joint Conference on Artificial Intelligence. Stockholm, Sweden (2018)
16. Zhu, Q., Wang, R., Chen, Q., Liu, Y., Qin, W.: Iot gateway: Bridging wireless sensor networks into internet of things. In: 2010 IEEE/IFIP International Conference on Embedded and Ubiquitous Computing. pp. 347–352 (December 2010)