On the Deployment of Factor Graph Elements to Operate Max-Sum in Dynamic Ambient Environments

P. Rust[1,2], G. Picard[2], and F. Ramparany[1]

[1] Orange Labs, France
{pierre.rust,fano.ramparany}@orange.com
[2] MINES Saint-Etienne, Laboratoire Hubert Curien UMR CNRS 5516, France
picard@emse.fr

Abstract. Using belief-propagation based algorithms like Max-Sum to solve distributed constraint optimization problems (DCOPs) requires deploying the factor graph elements on which the distributed solution operates. In some utility-based multi-agent settings, this deployment is straightforward. However, when the problem gains in complexity by adding other interaction constraints (like n-ary costs or dependencies), the question of deploying these shared factors arises. Here, we address this problem in the particular case of smart environment configuration (SECP), where several devices (e.g. smart light bulbs) have to coordinate as to reach an optimal configuration (e.g. find the most energy preserving configuration), under some n-ary constraints (e.g. physical models and user preferences). This factor graph deployment problem (FGDP) can be mapped to an optimization problem, then solvable in a centralized manner. But, when dealing with the dynamics of the environment (e.g. new sensed data which activates some rules, adding new devices, etc.) we cannot afford restarting the system or relying on a centralized solver. Thus, the system has to achieve on-line and local deployment adaptations. In this paper, we present some solutions and experiment them on a simulated smart home environment.

1 Introduction

A common problem when using distributed belief-propagation techniques as Max-Sum [7] is to decide where to host computations related to variable and factor message assessments. Indeed, Max-Sum operates on a factor graph which represents the problem to solve, by sending messages from variables to factors, and *vice versa*. Assessing messages to send requires computations to be hosted by some agents (or nodes). In some settings, this mapping is straightforward; this is the case in purely utility-based problems, where each agent owns a variable and a utility factor connected to some other variables owned by other agents [6]. However, in more complex settings such as interaction-based problems, where some factors and variables are shared by several agents, the question of deploying these elements arises.

An example of such settings is the smart environment configuration (SECP) we addressed in [11]. In SECP, several devices have to self-configure as to satisfy user requirements and to minimize energy consumption. SECP is modeled as a DCOP composed of n-ary factors corresponding to user rules (e.g. “setting the light level at 60

when someone is in the room"), physical models (e.g. "the effect of the light bulbs and the shutter on the light level in the room") and shared variables corresponding to physical properties (e.g. "the light level in the room"). Due to the dynamicity of the environment (e.g. devices appearing/disappearing, user adding/changing rules, sensed data updates, etc.), and the constrained communication and computation capabilities of our devices, deploying the factor graph is a key issue that cannot be solely solved off-line in a centralized manner.

The paper is structured along our contributions, as follows. Section 2 briefly exposes the SECP framework and its DCOP formulation. Section 3 introduces the factor graph deployment problem (FGDP) and presents an ILP to solve FGDP optimally in a centralized manner. The next sections discuss different cases of dynamics impacting the deployment: infrastructure changes (Section 4), problem and sensed environment changes (Section 5). Some preliminary experiments and related analysis are provided in Section 6. Finally, Section 7 concludes the paper.

2 Smart Environment Configuration

In this section, we expose the smart environment configuration problem we address in this paper, and some useful notations, as defined in [11].

Scenario. We consider the following AmI scenario. Our system is made of several smart devices (light bulbs, roller shutters, a TV set, etc.) and sensors (luminosity, presence, etc.). Each device is defined by (i) a unique identifier, (ii) its location (e.g. living room), (iii) a list of capabilities (e.g. emitting light or playing videos), (iv) a list of actions, (v) a consumption law that associates an energy cost to each action. The user can use an application on a tablet to configure simple behaviors (or *scenes*), using the value of the sensors or the state of actuators as triggers for implementing smart home actions. For example, one could configure the system such that a luminosity level of 60 is reached in the living room whenever somebody is in this room. The system autonomously decides the best way to achieve this targetDevices may be added or removed and are automatically integrated into the system. We want our system to choose the most energy-saving configuration for a given scene.

Problem Definition and Notations In [11], this configuration problem can be seen as an optimization problem with values to assign to actuators (e.g. a light bulb is assigned a power) and user's target values (e.g. the light level in living room is 60 lumens), while maximizing the adequacy to user-defined scenes and minimizing the overall energy consumption.

Problem 1 (SECP). Given a set of actuators $\mathfrak{A}$ (and their related costs $c_i \in \mathfrak{C}$), a set of sensors $\mathfrak{S}$, a set of scene rules $\mathfrak{R}$ (and their related utility functions in $u_k \in \mathfrak{U}$), and a set of physical dependency models Φ, the *Smart Environment Configuration Problem* (or *SECP*) $\langle \mathfrak{A}, \mathfrak{C}, \mathfrak{S}, \mathfrak{R}, \mathfrak{U}, \Phi \rangle$ amounts to finding the configuration of actuators that maximizes the utility of the user-defined rules, whilst minimizing the global energy consumption and fulfilling the physical dependencies.

Let $\mathfrak{A}$ the set of available actuators. We note $\nu(\mathfrak{A})$ the set of variables that represent the states of actuators $i \in \mathfrak{A}$ (e.g. the power assigned to a bulb). We use $\mathbf{x}_i$ to refer to a possible state of $x_i \in \nu(\mathfrak{A})$, that is $\mathbf{x}_i \in \mathcal{D}_{x_i}$ (domain of x_i). Activating an actuator i incurs a cost, noted $c_i : \mathcal{D}_{x_i} \rightarrow \mathbb{R}$, derived from the consumption law of each device. We note $\mathfrak{C} = \{c_i | i \in \mathfrak{A}\}$.

Let $\mathfrak{S}$ be the set of available sensors, and $\nu(\mathfrak{S})$ the set of variables encapsulating their states. We note $\mathbf{s}_\ell \in \mathcal{D}_{s_\ell}$ the current state of sensor $\ell \in \mathfrak{S}$. Sensor values are not controllable by the system: they are *read-only* values.

Let $\mathfrak{R}$ the set of user-defined scene rules. Each scene k is specified as a condition-action rule expressed using the set devices The condition part is specified as a conjunction of boolean expressions using state of actuators or sensors.The action part defines *target* values for either (i) some direct actions on actuators or (ii) indirect actions on abstract concepts (e.g. light level in living room) –both called *scene action variables*.

These scene action variables are therefore either (i) some $x_i \in \nu(\mathfrak{A})$ or (ii) other values constrained by values assigned to some actuators. We note $y_j \in \nu(\Phi)$ the state of such an *indirect* scene action j (e.g. the current level of light in a room), and $\mathbf{y}_j$ a possible state of y_j, that is $\mathbf{y}_j \in \mathcal{D}_{y_j}$. We note $\mathbf{x}_i^k$ (resp. $\mathbf{y}_j^k$) the target value defined by the user for the scene action variable x_i (resp. y_j) in the rule k. Obviously, $\mathbf{x}_i^k \in \mathcal{D}_{x_i}$ and $\mathbf{y}_j^k \in \mathcal{D}_{y_j}$ for all i, j and k. Note that a scene action variable can be used in several rules, but that a rule can only specify a unique target value for the scene action variable.

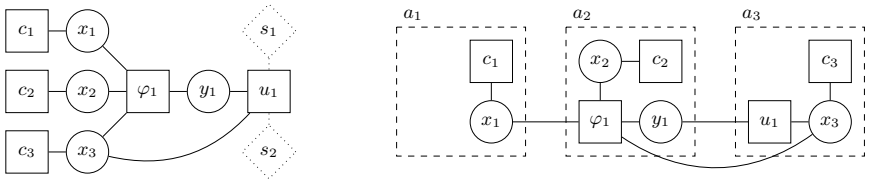
A scene rule can be either *active* or *inactive* depending on the state of devices appearing in the condition part of the rule. Each active scene has also a utility to be implemented, noted $u_k : \prod_{s \in \sigma(u_k)} \mathcal{D}_s \rightarrow \mathbb{R}$, with $\sigma(u_k) \subseteq \nu(\mathfrak{A}) \cup \nu(\Phi)$ being the scope of the rule (the subset of variables used in the rule). The more the states of the scene action variables (from $\nu(\mathfrak{A})$ and $\nu(\Phi)$) are close to the user's target values for this scene, the higher the utility. Moreover, if the condition to activate the rule (from $\nu(\mathfrak{A})$ and $\nu(\mathfrak{S})$) are not met, the utility should be neutral, i.e. equals to 0. We can therefore consider u_k's to be functions of the distance between the states of the scene action variables x_i's (resp. y_j's) and the target values $\mathbf{x}_i^k$ (resp. $\mathbf{y}_j^k$). We note $\mathfrak{U} = \{u_k | k \in \mathfrak{R}\}$.

Each scene action variable y_j depends physically on the values of several actuators. We note the model of this dependency $\phi_j : \prod_{\varsigma \in \sigma(\phi_j)} \mathcal{D}_\varsigma \rightarrow \mathcal{D}_{y_j}$, where $\sigma(\phi_j) \subseteq \nu(\mathfrak{A})$ is the scope of the model, i.e. the set of variables influencing y_j. Let $\Phi = \{\phi_j\}$ be the set of all physical models between actuators and user-defined values. In a more general form, a physical dependency model links a set of devices –with a given capability (e.g. emitting light, like a bulb or a TV set), in a given location (e.g. living room)– to a physical value (e.g. light level) that can be measured by some sensor (e.g. light sensor).

Formulation of SECP as a DCOP. SECP can be formulated as a DCOP $\langle \mathcal{A}, \mathcal{X}, \mathcal{D}, \mathcal{C}, \mu \rangle$ where: $\mathcal{A}$ is a set of smart devices; $\mathcal{X} = \nu(\mathfrak{A}) \cup \nu(\Phi)$; $\mathcal{D} = \{\mathcal{D}_{x_i} | x_i \in \nu(\mathfrak{A})\} \cup \{\mathcal{D}_{y_j} | y_j \in \nu(\Phi)\}$; $\mathcal{C} = \mathfrak{U} \cup \mathfrak{C} \cup \mathbf{\Phi}$; μ is a function that maps variables and constraints to smart devices; with the following objective:

$$\underset{\substack{x_i \in \nu(\mathfrak{A}) \\ y_j \in \nu(\Phi)}}{\textbf{maximize}} \ \omega_u \sum_{k \in \mathfrak{R}} u_k - \omega_c \sum_{i \in \mathfrak{A}} c_i + \sum_{\varphi_j \in \mathbf{\Phi}} \varphi_j \tag{1}$$

g



(a) Factor graph (b) Sample deployment

Fig. 1: Factor graph (1a) and a possible deployment (1b) on 3 nodes (a_1, a_2 and a_3)

Here, we note $\mathbf{\Phi}$ the corresponding set of φ_j's.

$$\varphi_j(x_j^1, \ldots, x_j^{|\sigma(\phi_j)|}, y_j) = \begin{cases} 0 & \text{if } \phi_j(x_j^1, \ldots, x_j^{|\sigma(\phi_j)|}) = y_j \\ -\infty & \text{otherwise} \end{cases} \tag{2}$$

SECP Factor Graph. Such a DCOP can be represented as a bipartite factor graph, noted $\mathcal{G} = \langle V_x, V_f, E \rangle$, which is a generalization of classical constraint graphs [7]. For SECP, variable nodes are taken from $V_x = \nu(\mathfrak{A}) \cup \nu(\mathit{\Phi})$, connected through factors in $V_f = \mathfrak{U} \cup \mathfrak{C} \cup \mathbf{\Phi}$ by applying the following rules: each $x_i \in \nu(\mathfrak{A})$ is a variable node, each $x_i \in \nu(\mathfrak{A})$ is connected to a unary factor c_i specifying its cost, each $y_j \in \nu(\mathit{\Phi})$ is a variable node, each y_j and all $x_i \in \nu(\mathfrak{A})$ in the scope of a physical dependency model ϕ_j are connected to a factor φ_j, each scene rule $k \in \mathfrak{R}$ is represented by a utility factor u_k connected to all the $x_i \in \sigma(u_k)$ and $y_j \in \sigma(u_k)$.

Example 1 (Factor graph). Fig. 1a represents a factor graph where x_1, x_2, x_3 are the state of light bulbs; c_1, c_2, c_3 are their activation costs ; u_1 is the factor representing the scene rule and defining the utility depending on a target value $\mathbf{y}_1^1$ for variable y_1; y_1 represents the theoretical light level in lumen; φ_1 is the physical dependency model between the light level and the state of actuators; s_1 and s_2 are read-only variable nodes, corresponding to sensor measurements, represented as dotted diamonds.

Variables and factors imply some computations on the hosting agents, depending on the size of the domains of the variables, the arity of the factors, and more generally on the complexity of the factors. Moreover, each link between two elements which are not hosted on the same agent implies some communication cost, depending on the size of the variable. Therefore, defining the mapping function μ which assigns each factor graph element to an agent is a key issue, called here the *deployment problem*.

Sensing-only devices run as *sleepy nodes*, meaning that they only turn their communication interface on when they want to emit a new value. As a result our FG is only hosted on actuator devices, as illustrated in Figure 1b. In the reminder, we use the terms "agents" and "nodes" interchangeably to denote these devices.

3 Optimal Deployment of Factor Graph Elements

As discussed in [11], the problem of deploying the factor graph elements on a set of nodes is equivalent to graph partitioning, which typically falls under the category of NP-hard problems [2,5]. Typically, such problem can be modeled as a mathematical optimization problem. To scale up, we propose here an integer linear program for general purpose, inspired by graph partitioning techniques from [5,3], and introduce some constraints which are specific to SECP.

Problem 2 (FGDP). Given a factor graph $FG = \langle V_x, V_f, E \rangle$ and a set of agents $\mathcal{A}$ the *Factor Graph Element Deployment Problem (FGDP)* amounts to assign each element of FG to an agent, while minimizing overall communications between agents.

First, we introduce some notations. As previously stated, μ is a function that maps elements of the FG (variables or factors) to nodes. We note $\mu_x^{-1}(a_k)$ (resp. $\mu_f^{-1}(a_k)$) the set of variables (resp. factors) hosted by agent a_k.

In SECP model it is usual that each actuator node has a computation capability. We consider variables $x_i \in \nu(\mathfrak{A})$ and constraints $f_j \in \mathfrak{C}$ related to each actuator to be *owned* by their actuator's node, meaning they will always be deployed on this specific node. We note $\rho(e) \in \mathcal{A} \cup \{\varnothing\}$ the owner of element $e \in V_x \cup V_f$, with $\phi(e) = \varnothing$ iff e does not belong to actuator node in $\mathcal{A}$ (i.e. $e \notin \mathfrak{C} \cup \nu(\mathfrak{A})$). We note $\rho_x^{-1}(a_k)$ (resp. $\rho_f^{-1}(a_k)$) the set of variables (resp. factors) owned by agent a_k. Remark that an owned element is always hosted on its owner, i.e. if $\rho(e) = a_k$ then $\mu(e) = a_k$

We note $\mathbf{com}(x_i, f_j)$ the communication load induced by the interaction between x_i and f_j. For instance, $\mathbf{com}(x_i, f_j)$ is the size of the messages exchanged between the variable and the factor:

$$\forall x_i \in V_x, f_j \in V_f, \quad \mathbf{com}(x_i, f_j) = \begin{cases} a.|\mathcal{D}_{x_i}| + b, \text{ if } (x_i, f_j) \in E \\ 0, \qquad\qquad \text{otherwise} \end{cases} \tag{3}$$

where a is the number of bytes to represent a value from the domain of variable x_i and b is the size of the message header. Let $\mathbf{mem}(e), e \in V_x \cup V_f$ be the memory footprint for the computation of factor graph element e. For instance, this is the size in bytes of the hypercube representing the costs in a factor. We also note $\mathbf{cap}(a_k)$ the memory capacity in bytes of node $a_k \in \mathcal{A}$.

Let's introduce the variables that map factor graph elements to agents, i.e. x_i^k (resp. f_j^k) denotes whether variable x_i (resp. factor f_j) is deployed in node a_k:

$$\forall x_i \in V_x, \quad x_i^k = \begin{cases} 1, \text{ if } \mu(x_i) = a_k \\ 0, \text{ otherwise} \end{cases} \tag{4}$$

$$\forall f_j \in V_f, \quad f_j^k = \begin{cases} 1, \text{ if } \mu(f_j) = a_k \\ 0, \text{ otherwise} \end{cases} \tag{5}$$

Moreover, for linearization purpose we introduce another set of variables (the α_{ijk}'s) which link variables to factors:

$$\forall x_i \in V_x, f_j \in V_f, a_k \in \mathcal{A}, \quad \alpha_{ijk} = x_i^k . f_j^k \tag{6}$$

Now, we are ready to model the factor graph element deployment problem (FGDP) as a linear program:

$$\underset{x_i^k, f_j^k}{\textbf{minimize}} \quad \sum_{(x_i,f_j)\in E} \sum_{a_k \in \mathcal{A}} \mathbf{com}(x_i, f_j).(1 - \alpha_{ijk}) \tag{7}$$

subject to

$$\forall x_i \in V_x, \quad \sum_{a_k \in \mathcal{A}} x_i^k = 1 \tag{8}$$

$$\forall f_j \in V_f, \quad \sum_{a_k \in \mathcal{A}} f_j^k = 1 \tag{9}$$

$$\forall a_k \in \mathcal{A}, \quad \sum_{x_i \in V_x} x_i^k + \sum_{f_j \in V_f} f_j^k \geq 1 \tag{10}$$

$$\forall (x_i, f_j) \in E, \quad \alpha_{ijk} \leq x_i^k \tag{11}$$

$$\forall (x_i, f_j) \in E, \quad \alpha_{ijk} \leq f_j^k \tag{12}$$

$$\forall (x_i, f_j) \in E, \quad \alpha_{ijk} \geq x_i^k + f_j^k - 1 \tag{13}$$

Objective (7) minimizes communications between factor graph elements which are not deployed on the same node. Constraints (8) and (9) force each factor graph element to be deployed on exactly one node. Constraint (10) enforces the use of all the available nodes. Finally, inspired by the linearization proposed in [3,1], constraints (11) to (13) link x_i^k's and f_j^k's to α_{ijk} in a linear way.

Problem 3 (ILP-FGDP). We term ILP-FGDP the 0/1 integer linear program consisting of objective (7) and constraints (8) to (13) which encodes the FGDP problem 2.

While Problem 3 is an ILP, and thus it is NP-hard, it can be solved in reasonable time in a centralized manner with branch-and-cut algorithm [10], especially when the coefficient matrix is sparse (which is our case here).

To take into account SECP specificities, we add the following constraints that will reduce the search space by ensuring that each owned element is hosted by its owner:

$$\forall a_k \in \mathcal{A}, \forall x_i \in \rho_x^{-1}(a_k), \quad x_i^k = 1 \tag{14}$$

$$\forall a_k \in \mathcal{A}, \forall f_j \in \rho_f^{-1}(a_k), \quad f_j^k = 1 \tag{15}$$

Finally, as SECP deals with devices with limited memory, we add a constraint to avoid memory capacity overflow:

$$\forall a_k \in \mathcal{A}, \quad \sum_{x_i \in V_x} \mathbf{mem}(x_i).x_i^k + \sum_{f_j \in V_f} \mathbf{mem}(f_j).f_j^k \leq \mathbf{cap}(a_k) \tag{16}$$

Problem 4 (ILP-SECP-FGDP). The integer linear program consisting of objective (7) and constraints (8) to (16) is an encoding of Problem 2 for SECP problems.

Problem 4 adds more constraints, but mainly restrict values for some variables, and thus strongly prune the search space. Thus, as proposed in [11], each time the user modify the factor graph by adding/removing/updating a rule on his interface, the optimal

deployment is computed and implemented. This can also be done in a fully distributed way by using a distributed simplex, as in [4]. However, computing the solution of this ILP in a limited computation node is not realistic. Therefore, we will discuss in the next section some techniques to repair deployments following some changes.

SECP has not been defined as a dynamic problem, but our implementation aims at optimizing an SECP instance each time a change occurs in the problem definition (e.g. the value of a sensor changes, which triggers a rule). The anytime nature of Max-Sum is very convenient for this setting, since each time a change occurs the involved variables and factors will send new messages. However, in the ambient dynamic and open environment we consider here, some issues due to dynamicity may arise, especially concerning the deployment of the factor graph on which Max-Sum operates.

4 Dynamics in the Infrastructure

We want to cope with changes in the infrastructure –i.e. the set of available agents/devices. Indeed, some questions arise: (i) how to manage factors and variables hosted by a device which disappeared? (ii) how to re-deploy factors and variables when a new device appears? One major constraint is that when such appearance and disappearance occur, the devices have to self-adapt without help of a central computer.

4.1 Notion of Neighborhood

Solving the whole ILP-SECP-FGDP problem for each device appearance and disappearance cannot be performed on one of the constrained devices the SECP is made of. Instead we consider adapting the deployment of the factor graph locally, by only considering a reduced set of agents (termed neighborhood) and a portion of the factor graph (set of elements hosted by the neighbors). Still, the solution to this problem may not be the global optimum w.r.t. ILP-SECP-FGDP, but potentially requires far less computation than solving the ILP-SECP-FGDP over the whole FG.

Let's define the notion of neighborhood as follows:

Definition 1. *Given the current assignment μ, the* neighborhood *of an agent a_k is defined as follows: $\mathcal{A}[a_k] = \{a_\ell \mid \exists (x_i, f_j) \in E, \mu(x_i) = a_k, \mu(f_j) = a_\ell\} \cup \{a_\ell \mid \exists (x_i, f_j) \in E, \mu(f_j) = a_k, \mu(x_i) = a_\ell\} \cup \{a_k\}$, if the agent a_k hosts at least one FG element, and $\mathcal{A}[a_k] = \mathcal{A}$ otherwise.*

Similarly we define the set of edges connected to the neighborhood as $E[a_k] = \{(x_i, f_j) \mid \mu(x_i), \mu(f_j) \in \mathcal{A}[a_k]\}$ and the set of neighborhood variables (resp. factors) as $V_x[a_k] = \{x_i \mid (x_i, f_j) \in E[a_k]\}$ (resp. $V_f[a_k] = \{f_j \mid (x_i, f_j) \in E[a_k]\}$).

4.2 Adaptation to device arrival

Here the initial deployment should be revised as to benefit from new computation and memory capacities. However, since our devices have limited computation and memory capacities, we cannot afford solving again Problem 4. Thus we propose here to restrict the revision of the factor graph to the neighborhood of the newcomer.

There are two situations to consider:

(i) The device is an *actuator* owning, and thus already hosting, variables and factors (e.g. adding a light bulb with its decision variable and cost factor). The new device, depending on its capabilities (e.g. emitting light in room #1) has to connect to corresponding physical factors (e.g. light model for room #1) and/or rule factors (e.g. a rule switching on all light bulb in room #1)[3]. If we simply add the newcomer to the system, the resulting factor graph deployment will not be optimal in the sense of Problem 4 and could generally be improved by migrating some elements as to reduce communication costs.

(ii) The device is a *non-actuator* which only provides computation and memory, without already hosting variables nor factors. In order to benefit from these capabilities, existing elements must be relocated to the newcomer. In this case, the redeployment process amounts to selecting the elements to migrate as to optimize communication costs.

We analyse here two mechanisms, that can be used for both cases.

Restricted ILP-SECP-FGDP The idea here is to encode the deployment revision problem as a cut version of Problem 4, restricted to the neighborhood of the newcomer. For each device in this neighborhood, the problem consists in choosing the elements to host, with respect to communication and memory capacities.

Problem 5 (ILP-FGDP$[a_k]^+$). ILP-SECP-LGDP$[a_k]^+$ consists in ILP-SECP-LGDP (4) restricted to the set of agents $\mathcal{A}[a_k]$ and to factor graph $\langle V_x[a_k], V_f[a_k], E[a_k]\rangle$.

This problem can be solved either by one agent (if the size of the problem is not too large) or by the agents composing the neighborhood. In both case it only requires local and limited knowledge on the global DCOP, which makes it ideal for large and complex systems. Prior to solving this problem, agents have to share their elements and costs with other agents involved in the revision. The worst case, when a newcomer is connected to all other agents, is equivalent to solve ILP-SECP-FGDP on the whole FG, which may not be reasonable in terms of response time and communication load. In this case one could devise a method to select another subset of agents as a neighborhood to solve ILP-SECP-LGDP$[a_k]^+$ on.

In the distributed solving case, several distributed optimization techniques could meet the requirements like the distributed simplex method designed for multi-agent assignments [4], keeping exactly the same encoding than ILP-SECP-FGDP, or dual decomposition methods like the efficient AD^3 method [9], that requires ILP-SECP-FGDP to be encoded using tractable high order potentials [12], and then implement a distributed decoding of the LP relaxation to assign integer values to decision variables. However, while providing good optimality, both distributed simplex and AD^3 may require several rounds (thus message exchanges) to reach good quality solutions. For instance, from a conjecture in [4], the average time complexity of this technique is linear in the diameter of the graph ($\mathcal{O}(diam(FG))$), with polynomial communication load. In SECP case, the diameter of the FG is not bounded but mainly depends on

[3] This discovery phase is not discussed in this paper.

the number of rules and models, and their interdependencies. In the case of real smart home settings, models and rules will mostly influence local areas (rooms, floor, etc.) and interdependencies, thus diameters, will be limited.

Newcomer Decision problem. As to avoid high communication load induced by the previous techniques, we can consider a more newcomer-centric approach: the newcomer calls for proposals to move some computations. Based on the costs of the proposed computations and its own memory capacity, the newcomer has to choose a set of factor graph elements to host. Let's formulate this newcomer decision problem.

Problem 6 (NDP). Given a newcoming agent and a set of proposed computations to migrate coming from its neighborhood, the *Newcomer Decision Problem* (NDP) amounts to choose computations amongst proposed computations, so that communication load is minimized and memory constraints are fulfilled.

Each neighbor $a_\ell \in \mathcal{A}[a_k]$ sends its proposal in message $\langle V^{\ell\to k}, E^{\ell\to k}, \mathbf{com}\rangle$, where: $V^{\ell\to k} \subset V_x \cup V_f$ is the set of elements (factors and variables) it proposes; $E^{\ell\to k} = \{(e_i, e_j) \mid (e_i, e_j) \in E, e_i \in V^{\ell\to k} \text{ or } e_j \in V^{\ell\to k}\}$ is the set of edges connected to elements in $V^{\ell\to k}$; and $\mathbf{com}$ is the communication cost function (potentially restricted to elements in $E^{\ell\to k}$). We note $V^k = \bigcup_\ell V^{\ell\to k}$ and $E^k = \bigcup_\ell E^{\ell\to k}$. We assume the communication cost $\mathbf{com}(e_i, e_j)$ can be assessed only using information sent by proposers. Let e_i^k be a binary variable stating whether the newcomer a_k chooses to host computation e_i. The cost of selecting a set of computations can be formulated as follows:

$$\sum_{(e_i,e_j)\in E^k} \mathbf{com}(e_i, e_j)(e_i^k + e_j^k - 2.e_i^k.e_j^k) \tag{17}$$

$$-\sum_{(e_i,e_j)\in E^k} \mathbf{com}(e_i, e_j).e_i^k.e_j^k \tag{18}$$

which is composed of the sum of the communication costs for the set of edges which are cut in the new distribution (17), i.e. those for which e_i^k XOR e_j^k holds true; minored by the communication costs for the set of edges whose both ends are now hosted on the same agent (18), i.e. those for which x_i^k AND e_j^k holds true. This sum can be simplified and used as the optimization objective for the newcomer a_k, as follows:

$$\underset{e_i^k, e_j^k}{\textbf{minimize}} \sum_{(e_i,e_j)\in E^k} \mathbf{com}(e_i, e_j)(e_i^k + e_j^k - 3.e_i^k.e_j^k) \tag{19}$$

$$\textbf{subject to} \quad \sum_{e_i\in V^k} \mathbf{mem}(e_i).e_i^k \leq \mathbf{cap}(a_k) \tag{20}$$

Problem 7 (IQP-NDP). We term IQP-NDP the $0/1$ integer quadratic program consisting of quadratic objective (19) and linear constraints (20) which encodes NDP.

This problem falls into the quadratic knapsack problem (QKP) framework. Indeed, Equation (19) can be reformulated as follows:

$$\underset{e_i^k, e_j^k}{\textbf{minimize}} \sum_{e_i\in V^k} e_i^k.\mathbf{p}(e_i) + \sum_{\substack{e_i\in V^k \\ e_j\in V^k}} e_i.e_j.\mathbf{P}(e_i, e_j) \tag{21}$$

with
$$\mathbf{p}(e_i) = \sum_{e_j \in V^{k+}} \mathbf{com}(e_i, e_j), \quad \forall e_i \in V^k \tag{22}$$

$$\mathbf{P}(e_i, e_j) = \begin{cases} -3.\mathbf{com}(e_i, e_j), & \text{if } (e_i, e_j) \in E^k \\ 0, & \text{otherwise} \end{cases} \tag{23}$$

and $V^{k+} = \{e_i \mid (e_i, e_j) \in E^k \text{ or } (e_j, e_i) \in E^k\}$ is the set of elements connected to at least one edge in E^k, even the ones that are not movable (thus, not necessarily proposed for migration).

QKP can be linearized [1] and then solved using a centralized branch-and-cut method or a distributed optimization method, as discussed earlier. Alternatively, QKP is solvable by dynamic programming, but without optimality guarantees. Only requiring $\mathcal{O}(\mathbf{cap}(a_k).|V^k|)$ space, and $\mathcal{O}(\mathbf{cap}(a_k).|V^k|^2)$ time, such a dynamic programming approach seems realistic in our case [8].

Besides, instead of using its whole memory capacity $\mathbf{cap}(a_k)$, device a_k may also set a limit capacity below its maximum one (e.g. the average memory used by its neighbors) as not to host more computation than others, in general. As to respect constraint (10) in Problem 4, we can add the following constraints to a_k's decision to enforce hosting at least one element:

$$|\rho^{-1}(a_k)| + \sum_{e_i \in V^k} e_i^k \geq 1 \tag{24}$$

$$\forall a_\ell \in \mathcal{A}[a_k] \setminus a_k, \ |\mu^{-1}(a_\ell)| - \sum_{e_i \in V^{\ell \to k}} e_i^k \geq 1 \tag{25}$$

Constraint (24) ensures that the newcomer hosts at least one computation. Constraint (25) avoids migrating all elements from one of the proposing agents and requires the newcomer to known the number of elements hosted by a_ℓ.

From the proposer side, the decision of choosing which elements to propose is also an issue, that may impact the newcomer's decision. In this paper, we only consider proposing all the "movable" elements, i.e. those that are hosted but not owned by the agent (physical factors and variables, and rules utility factors).

4.3 Device Removal

Another common problem in ambient environments is that some devices may fail or get unreachable for some reason. In this case, for belief-propagation algorithms to operate properly, we need to fix the deployment: factor and variables *owned* by the departed agent simply disappear from the factor graph while elements *hosted* on it must be relocated to an available agent. We can identify two cases: *safe* removal (the device leaves the system voluntarily and can migrate elements before leaving), and *unsafe* removal (when the device fails without migrating its hosted elements).

Safe removal is equivalent to the device arrival, but the element allocation is made over the neighborhood of the removed agents (excluded). Here, solving IQP-NDP, is not relevant, because no agent is at the center of the decision. We will not elaborate on this case.

Unsafe removal is more complex, and implies some technicalities. Here, we assume that devices are aware of disappearance of any device from their neighborhood (using *keepalive* signals). In such case, one could solve ILP-SECP-FGDP for the entire new

set of devices, which we cannot afford within a single device. Therefore, we opt here for a more local and heuristic approach. The idea is to solve ILP-SECP-FGDP restricted to agents that were directly connected to the dead device.

We note $V_x[a_k]^- = V_x[a_k] \setminus \rho_x^{-1}(a_k)$, $V_f[a_k]^- = V_f[a_k] \setminus \rho_f^{-1}(a_k)$ and $E[a_k]^- = E[a_k] \cap (V_x[a_k]^- \times V_f[a_k]^-)$ the sets of elements and edges involved in the redistribution. We note $\mathbf{cap}^-(a_k) = \mathbf{cap}(a_k) - \sum_{e_i \in \rho^{-1}(a_k)} \mathbf{mem}(e_i)$ the memory capacity of any a_k, obtained by subtracting from $\mathbf{cap}(a_k)$ the memory footprint of computations hosted on a_k and not involved in the redistribution.

Problem 8 (ILP-SECP-FGDP$[a_k]^-$). ILP-SECP-FGDP$[a_k]^-$ consists in ILP-SECP-FGDP restricted to the set of agents $\mathcal{A}[a_k] \setminus \{a_k\}$ and the factor graph $\langle V_x[a_k]^-, V_f[a_k]^-, E[a_k]^- \rangle$, and where $\mathbf{cap}$ is replaced by $\mathbf{cap}^-$.

To solve ILP-FGDP$[a_k]^-$ some requirements are to be fulfilled: (a) devices have to know how to compute elements which share an edge with an element they host and the corresponding communication costs; (b) devices have to know to which devices they send messages; (c) devices need to have enough memory to host new elements.

Requirements (a) imply giving such information during initial deployment and revision phases. Requirement (b) depends upon the discovery mechanism when an agent hosts new elements, for example when new devices are added. Finally, requirement (c) may not be reached if the neighboring agents of a disappearing one don't have enough memory all together. In this case, the neighborhood can be extended by neighbors of neighbors until memory is sufficient. Once these requirements met, agents can solve ILP-SECP-FGDP$[a_k]^-$ on the limited set of elements and the neighboring devices, as it is the case for a newcoming device (see Section 4.2).

5 Discussion on Other Dynamics

Up to now, we have discussed dynamics involved by adding or removing devices. However, even for a same set of devices, other dynamics may occur, since our socio-technical system is both connected to users and a sensed environment.

Changing SECP Elements The first reason to alter the deployed FG, is for a user to add or remove a rule to the system. This implies deploying new factors and variables in the current infrastructure. However, in this case, since the user is interacting with the system through a dedicated device (e.g. home computer, tablet) and not a limited one, this is equivalent to deploying the initial FG, as in Section 3.

The second reason to alter the FG is to add/remove actuator variables and costs. But, this only occurs when adding/removing devices, as in Sections 4.2 and 4.3.

The third reason to alter the FG, is to add new physical models. This is the most difficult part. Indeed, here we assume these physical models are provided (either by a manufacture setting, or following a calibration phase during the installation). Adding a new physical model only makes sense when the user specifies a new rule with a new physical model (related to a new sensor) he has obtained. For instance, a user adds a sound level sensor in his apartment and add a new rule which exploits the sound level

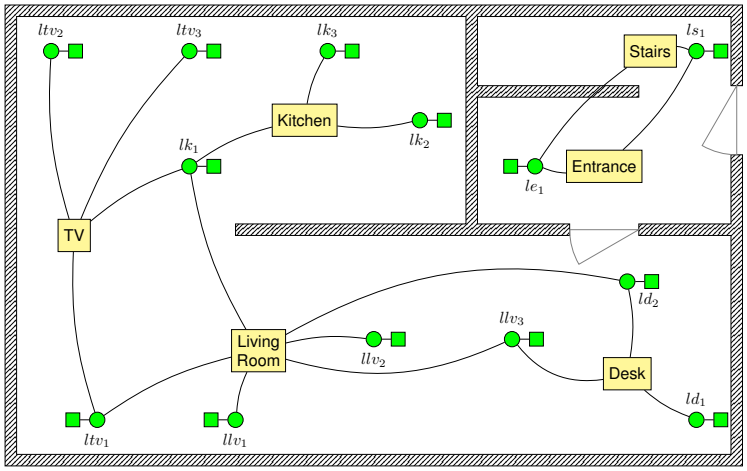


Fig. 2: Map of a simulated smart home, and corresponding initial physical models, actuators and costs.

somehow. Such a situation, once again, only occurs when the user interacts through his dedicated device with the system. Thus, deploying the FG can be done in a centralized way, as in Section 3.

Changing Factors Following a Environmental Change. Our system is deployed in a dynamic environment, where newly sensed data may imply that some rules activates or not. Up to now, we discussed the deployment of the whole FG, but practically, all rules are not necessarily active all the time. Rules, and therefore some factors and variables are only active when some sensed state is reached. Such activation/deactivation may greatly impact the performance of the system, by adding/removing computations and loops in the factor graph. Our deployment model does not take this into account. In fact, it considers the worst case when all the elements are active.

6 Experiments

As to evaluate the performances of the proposed repair techniques, we simulate a smart home where devices perform Max-Sum as to find the optimal configuration considering user preferences and energy consumption, inspired by the used case proposed in [11].

In our simulations, two types of events may occur: device arrival (in) and unsafe device removal (out). In case of device arrival, we use either ILP-FGDP$[a_k]^+$, which is solved using a classical ILP solver within one node (using GLPK in our simulator), or IQP-NDP, which is solved using a dedicated dynamic program (embedded in our simulator, in Python)[4], both defined in Section 4.2. In case of device removal, as discussed

[4] Such discrepancies in terms of solution method implementation are the reason not to plot computation times

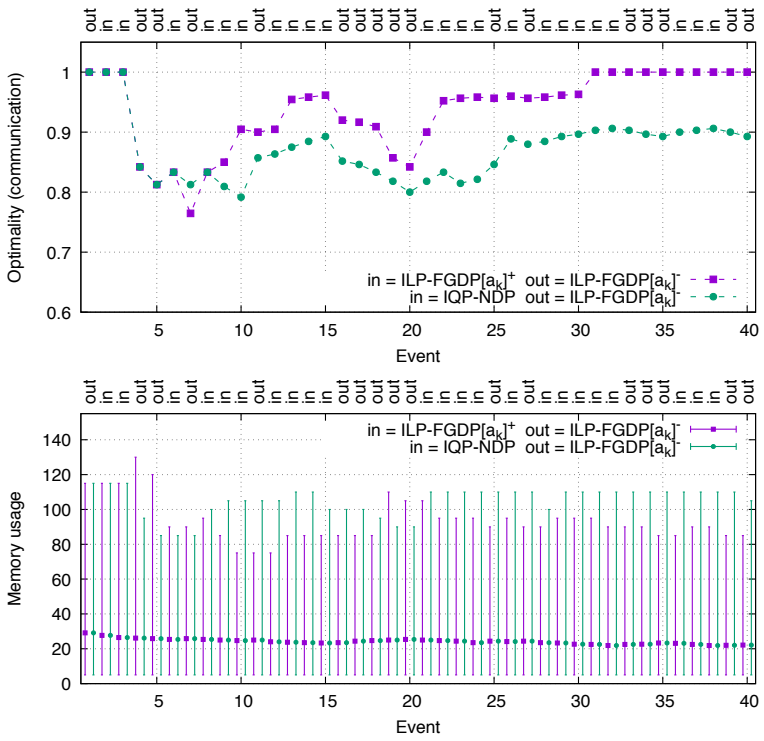


Fig. 3: Optimality (3a), and the memory usage (3b) of the deployment during the simulation (average, min and max).

in Section 4.3, ILP-FGDP$[a_k]^-$ is solved using GLPK. Whatever the type of event, the best ILP-FGDP solution is also computed with GLPK, to benchmark aforementioned methods.

In a first series of experiments, we simulate the first floor of a smart home, as represented in Figure 2, which is initially composed of 13 actuators (light bulbs and their respective costs), 6 physical models (one for each space), and 5 user rules (not represented in Figure 2, for clarity). For communication costs, $a = 5$, $b = 100$, and the default agent memory capacity ($\mathbf{cap}$) is set to 200 memory units (one unit represents the space to store one value, e.g. 32 bits). Figure 3 traces performances of repair solutions on a scripted scenario where devices are added and removed at runtime. Each of these 40 events is followed by a repair phase using the proposed methods. Figure 3a shows the optimality of the repaired deployments, which is computed as the ratio between the repaired cost and the best cost (real ILP-FGDP optimum). Clearly, with both approaches, out events tends to degrade the optimality of the deployment, while still maintaining it at a very competitive level, compared to a full deployment of the whole factor graph. Interestingly, in events improve optimality, meaning that in real systems where on average out are approximately balanced by in, the deployment should keep a very good quality level. Figure 3b presents the average (and min and max) memory usage over all

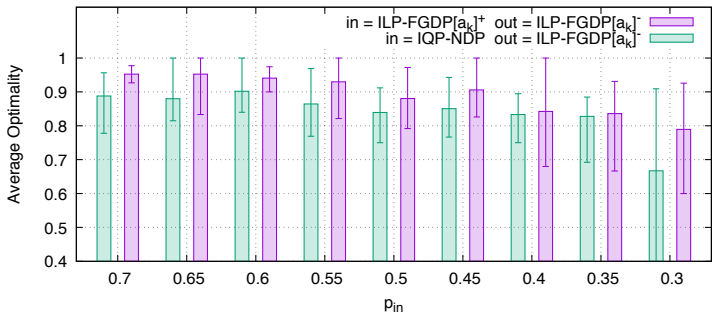


Fig. 4: Influence of the p_{in} probability on the optimality

the devices, after each event. While our approaches are not specifically designed to ensure a fair memory load share among devices, both methods do not lead to an excessive accumulation of computations on a single device. Solving ILP-FGDP$[a_k]^+$ is clearly a better choice in this regard, which can be explained by the fact that it allows relocation of computation on the full neighborhood, while solving IQP-NDP only allows migration of computations to the newcomer.

In a second series of experiments, we simulate the whole house with 23 actuators, 9 physical models and 9 rules. Here we evaluate the robustness of each repair techniques with more and more device removal. Figure 4 shows the average performances over 10 simulations after 20 events, in terms of optimality (computed as previously) with a varying event type probability. At each event generation, its type is determined using p_{in}, i.e. the probability for en event to be in. The higher p_{in}, the easier the adaptation is, since more devices are probably added. ILP-FGDP$[a_k]^+$ combined with ILP-FGDP$[a_k]^-$ presents very good resilience, since it offers more than 80% optimality with $p_{\text{in}} \geq 0.35$ (approx. 2 removals for 1 arrival). IQP-NDP combined with ILP-FGDP$[a_k]^-$ is always 5 to 15% lower.

Finally, ILP-FGDP$[a_k]^+$ presents better optimality, but requires much more information to be computed, whilst IQP-NDP is in average 10% worse in communication cost, and equivalent in average memory usage.

7 Conclusions

In this paper we discussed and analyzed the problem of deploying factor graph elements within a open infrastructure composed of constrained devices. We model the deployment problem as a graph partitioning problem, encoded as a binary integer linear problem, to be solved each time the user pushes new rules in the system. We also discussed several repair techniques to cope with device arrival and removal occurring at runtime, by solving the original deployment problem on a restricted set of devices and factor graph elements, or implementing a newcomer-centric approach. Experiments we made on a simulated environment show that the proposed local and heuristic techniques have competitive optimality levels in comparison to restarting the deployment from scratch.

Additionally, these techniques only use limited and local knowledge and thus could be used in arbitrarily large systems. As mentioned when dealing with newcoming agents, the decision of choosing which elements to propose is also an issue we did not investigate. Here, we might consider basing agents' decisions on preferences or history of past computations and messages exchange, as to assess elements to send. Besides, we didn't discuss the update of physical models following the appearance/disappearance of devices. We let this problem, related to machine learning, to future research.

References

1. Adams, W.P., Forrester, R.J., Glover, F.W.: Comparisons and enhancement strategies for linearizing mixed 0-1 quadratic programs. Discrete Optimization 1(2), 99 – 120 (2004), //www.sciencedirect.com/science/article/pii/S1572528604000210
2. Bichot, C.E., Siarry, P. (eds.): Graph Partitioning. Wiley (2011)
3. Boulle, M.: Compact mathematical formulation for graph partitioning. Optimization and Engineering 5(3), 315–333 (2004), http://dx.doi.org/10.1023/B:OPTE.0000038889.84284.c7
4. Bürger, M., Notarstefano, G., Bullo, F., Allgöwer, F.: A distributed simplex algorithm for degenerate linear programs and multi-agent assignments. Automatica 48(9), 2298 – 2304 (2012), //www.sciencedirect.com/science/article/pii/S0005109812002956
5. Fan, N., Pardalos, P.M.: Linear and quadratic programming approaches for the general graph partitioning problem. Journal of Global Optimization 48(1), 57–71 (2010), http://dx.doi.org/10.1007/s10898-009-9520-1
6. Farinelli, A., Rogers, A., Jennings, N.R.: Agent-based decentralised coordination for sensor networks using the max-sum algorithm. Autonomous Agents and Multi-Agent Systems 28(3), 337–380 (May 2014), http://link.springer.com/10.1007/s10458-013-9225-1
7. Farinelli, A., Rogers, A., Petcu, A., Jennings, N.R.: Decentralised coordination of low-power embedded devices using the max-sum algorithm. In: International Conference on Autonomous Agents and Multiagent Systems (AAMAS'08). pp. 639–646 (2008), http://dl.acm.org/citation.cfm?id=1402298.1402313
8. Fomeni, F.D., Letchford, A.N.: A dynamic programming heuristic for the quadratic knapsack problem. INFORMS Journal on Computing 26(1), 173–182 (2014), http://dx.doi.org/10.1287/ijoc.2013.0555
9. Martins, A.F.T., Figueiredo, M.A.T., Aguiar, P.M.Q., Smith, N.A., Xing, E.P.: Ad3: Alternating directions dual decomposition for map inference in graphical models. Journal of Machine Learning Research 16, 495–545 (2015), http://jmlr.org/papers/v16/martins15a.html
10. Mitchell, J.E.: Handbook of Applied Optimization, chap. Branch-and-Cut Algorithms for Combinatorial Optimization Problems, pp. 65–77. Oxford University Press (2002)
11. Rust, P., Picard, G., Ramparany, F.: Using message-passing DCOP algorithms to solve energy-efficient smart environment configuration problems. In: International Joint Conference on Artificial Intelligence (IJCAI). AAAI Press (2016)
12. Tarlow, D., Givoni, I.E., Zemel, R.S.: Hop-map: Efficient message passing with high order potentials. In: Teh, Y.W., Titterington, D.M. (eds.) Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics (AISTATS-10). vol. 9, pp. 812–819 (2010), http://www.jmlr.org/proceedings/papers/v9/tarlow10a/tarlow10a.pdf