POLITECNICO DI TORINO
Repository ISTITUZIONALE

EUDoptimizer: Assisting End Users in Composing IF-THEN Rules Through Optimization

Original
EUDoptimizer: Assisting End Users in Composing IF-THEN Rules Through Optimization / Corno, F., De Russis, L., Monge Roffarello, A.. - In: IEEE ACCESS. - ISSN 2169-3536. - ELETTRONICO. - 7:(2019), pp. 37950-37960. [10.1109/ACCESS.2019.2905619]

Availability:
This version is available at: 11583/2728803 since: 2019-04-04T09:18:22Z

Publisher:
IEEE

Published
DOI:10.1109/ACCESS.2019.2905619

Terms of use:

This article is made available under terms and conditions as specified in the corresponding bibliographic description in the repository

Publisher copyright

(Article begins on next page)

30 October 2026

IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received February 7, 2019, accepted March 12, 2019, date of publication March 18, 2019, date of current version April 5, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2905619

EUDoptimizer: Assisting End Users in Composing IF-THEN Rules Through Optimization

FULVIO CORNO, (Member, IEEE), LUIGI DE RUSSIS, (Member, IEEE), AND ALBERTO MONGE ROFFARELLO, (Student Member, IEEE)
Dipartimento di Automatica e Informatica, Politecnico di Torino, 10129 Turin, Italy
Corresponding author: Luigi De Russis (luigi.derussis@polito.it)

ABSTRACT Nowadays, several interfaces for end-user development (EUD) empower end users to jointly program the behavior of their smart devices and online services, typically through trigger-action rules. Despite their popularity, such interfaces often expose too much functionality and force the user to search among a large number of supported technologies disposed of confused grid menus. This paper contributes to the EUD with the aim of interactively assisting end users in composing IF–THEN rules with an optimizer in the loop. The goal, in particular, is to automatically redesign the layout of the EUD interfaces to facilitate the users in defining triggers and actions. For this purpose, we define a predictive model to characterize the composition of trigger-action rules on the basis of their final functionality, we adopt different optimization algorithms to explore the design space, and c) we present *EUDoptimizer*, the integration of our approach in IFTTT, one of the most popular EUD interfaces. We demonstrate that good layout solutions can be obtained in a reasonable amount of time. Furthermore, an empirical evaluation with 12 end users shows evidence that *EUDoptimizer* reduces the efforts needed to compose trigger-action rules.

INDEX TERMS Combinatorial optimization, end-user development, service automation, trigger-action programming.

I. INTRODUCTION

The need of providing end users with adequate paradigms and tools to customize their smart environments on the basis of their personal needs has recently gained interest [1]. Nowadays, in fact, end users interact with a huge number of smart devices and online services on a daily base. End-User Development (EUD) methodologies [2] are suitable for letting non-technical users to program their smart objects and web services. In this field, the most commonly adopted EUD paradigm is trigger-action programming, with which users can define IF-THEN rules such as *"if I receive a Facebook notification, then blink the Philips Hue lamp in the kitchen"* or *"if the Nest camera detects a movement, then set 22 degrees on my Nest thermostat."*

Contemporary EUD interfaces, e.g., IFTTT[1] and Zapier[2], are typically organized through grid layouts (Figure 1), i.e., a particular type of menu[3] where items are organized in rows and columns. Furthermore, they share the same modality for composing trigger-action rules [1]: users have to firstly select a *technology* from a grid menu. The technology represents the manufacturer or the brand of a supported smart device or web service, e.g., *Philips Hue* or *Facebook*. On the selected technology, users can then define a particular trigger (*if*) through a wizard-based procedure. The same operations need to be repeated to define the action (*then*) of the rule.

Despite apparent simplicity, the composition of trigger-action rules through EUD interfaces is challenging, mainly because the rapid spread of new heterogeneous technologies [3]. Contemporary EUD interfaces display a huge number of information without any support to discover useful triggers and actions, and become too complex for non programmers. Consequently, users without technical skills may not find these systems useful [4]. IFTTT and Zapier, for example, force users to define their rules by searching between 500 and 1,000 supported technologies displayed with a meaningless order (Figure 1). Many previous works tried to mitigate such a complexity in different ways, thus confirming the need of better assisting users in programming their devices and services. Some of them, for example,

The associate editor coordinating the review of this manuscript and approving it for publication was Shiqiang Wang.

[1] https://ifttt.com/ last visited on January 12, 2018

[2] https://zapier.com/ last visited on January 12, 2018

[3] In the remainder of the paper, grid layout and grid menu will be used interchangeably

This work is licensed under a Creative Commons Attribution 3.0 License. For more information, see http://creativecommons.org/licenses/by/3.0/

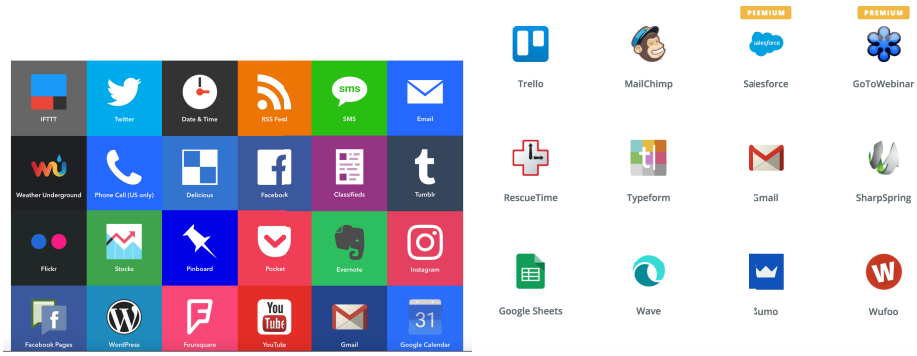


FIGURE 1. Selection of a technology for defining triggers (or actions) in IFTTT (a) and Zapier (b). In each platform there are, as of today, between 500 and 1,000 supported devices and online services.

face the problem by changing the underlying representations [5], [6], while others by exploring new composition paradigms, e.g., [1].

In this paper, we explore a different approach that is fully compatible with contemporary EUD interfaces, such as those of Figure 1, thus not requiring any radical change. Rather than acting on representations or composition paradigms, we adopt an optimizer in the loop to interactively assist end users in composing IF-THEN rules. The goal, in particular, is to *dynamically* redesign grid layouts in EUD interfaces in an *interactive* way, i.e., by considering the choices made by end users during the rule composition phase.

To reach our goal, we define two different predictive models to be used in a multi-objective optimization problem. In particular, we adapt Search-Decision-Pointing (SDP), a state-of-the-art predictive model of user performance in linear menu search, to work with grid layouts. Furthermore, we propose the Functionality Similarity Model (FSM), a novel model based on Semantic Web to take into account whether different and heterogeneous technologies provide similar functionality. As previous works demonstrate [7], [8], we claim that users would benefit from more support in discovering functionality, rather than being forced to get acquainted with technological details. Users, in fact, are often interested in what a device or service can do, and they reason about abstract behaviors, e.g., *"turn on the lights of the kitchen"*, rather than specific technologies, e.g., *"turn on the Philips Hue lamp in the kitchen."* To explore the design space of grid-based EUD interfaces, we consider two different optimization algorithms, i.e., Simulated Annealing and Ant Colony Optimization, and we integrate those optimization methods in *EUDoptimizer* (***E**nd-**U**ser **D**evelopment **optimizer***), an implementation of our approach on top of IFTTT. The implementation allowed us to qualitatively and quantitatively evaluate our approach. By exploiting a dataset of more than 200,000 trigger-action rules extracted from IFTTT [7], we show that *EUDoptimizer* can produce satisfactory solutions in a reasonable amount of time. Moreover, data from a user study with 12 participants suggest that *EUDoptimizer* can help end users define IF-THEN rules with less time and cognitive effort.

The paper is organized as follows. In Section II we contextualize our work by describing related works. In Section III we exemplify our approach by presenting a use case of *EUDoptimizer*. Then, we describe the components needed to realize such a use case, i.e., the predictive models (Section IV), the optimization methods (Section V), and the implementation of the EUDoptimizer tool that allows the interaction between users and the optimization method (Section VI). Section VII and Section VIII report a technical assessment of the implemented algorithm and the evaluation of *EUDoptimizer* in a user study, respectively. Eventually, Section IX discusses results and Section X concludes the paper.

II. BACKGROUND AND RELATED WORK

In this section, we first report some previous works that aim at improving EUD approaches and interfaces. Then, we contextualize our work in the large topic of applying optimization methods to user interface design.

A. IMPROVING END-USER DEVELOPMENT

Lieberman *et al.* [2] define End-User Development as *"a set of methods, techniques, and tools that allow users of software systems, who are acting as non-professional software developers, at some point to create, modify or extend a software artifact."* With the technological advances we are confronting today, people are increasingly moving from passive consumers to active producers of information, data, and software: in the last 10 years, several commercial tools for end-user personalization of devices and services, such as IFTTT or Zapier, were born.

Some previous works try to overcome the issues characterizing such tools, e.g., complexity [4] and scarce expressiveness [7], by acting on the underlying models, with the aim of simplifying the trigger-action rules composition process. Barricelli and Valtolina [5] present an extension of the trigger-action paradigm to better cope with the evolving Internet of

Things (IoT) scenario. Besides devices and Web applications, they also incorporate other users, space and time, and the social dimension. With the EUPont ontology, Corno *et al.* [6] propose a high-level ontology for modeling triggers and actions. The semantic model allows the definition of generic rules that can be adapted for different contextual situations. In our work, we use EUPont in the predictive model used for assessing whether different technologies provide similar functionality.

Other works focus on new interfaces and tools for End-User Development. Desolda *et al.* [1], for example, report the results of a study to identify possible visual paradigms to compose trigger-action rules for smart environments, and present the architecture of a platform to support rules execution.

In our work, we follow a different approach. We are not interested in changing the underlying models nor the composition paradigms. To assist end users in defining IF-THEN rules, we employ optimization methods to dynamically redesign the grid menus of contemporary EUD interfaces. In a certain way, our approach can be seen as a sort of recommender system. In fact, by exploiting models of user performance and perception, we *suggest* technologies to be used in triggers and actions to the user, dynamically.

B. COMBINATORIAL OPTIMIZATION OF USER INTERFACES

Applying optimization methods to user interface design is a long standing topic in human-computer interaction research: when assumptions are appropriate, optimization methods offer a greater-than-zero chance of finding an optimal design [9]. Optimization methods have been firstly adopted for keyboard layouts [10], [11], and then in many other application areas, e.g., accessibility [12], menus [13], [14], sketching [15], and web layouts [16]. Since EUD interfaces are typically organized through grid menus, our work is strictly related to the menu optimization problem. Due to large design space, menus are good candidates for optimization problems. A menu with n elements, in fact, can be organized in $n!$ ways, and design heuristics, e.g., displaying frequently used items at the top, may be effective for small n but fail with larger n or if additional human factors such as semantic relationships among items are considered [14]. Combinatorial optimization methods, instead, explore a large number of designs in order to find a good (preferably optimal) solution that minimize or maximize an objective function. While the computational cost is often a problem, reasonable solutions can be obtained by adopting *interactive* approaches, i.e., by involving users in redefining and refining the optimization problem [14]. In our work, the problem of defining trigger-action rules contains steps that are intrinsically interactive: components layout for defining an action, for example, may change on the basis of the defined trigger.

Similarly to previous works , we follow a model-based optimization approach [9]. Unlike heuristic approaches, which do not predict effects on end users, model-based optimization exploits predictive models of user performance and layout perception. The idea is to represent a design problem, along with a design knowledge, as an objective function. Then, a search algorithm is used to iteratively improve designs for the stated object. Several models of user performance and layout perception have been proposed for specific tasks. Examples can be seen in Sketchplore [15], a multi-touch sketching tool that uses a real-time layout optimizer, and MenuOptimizer [14], an interactive design tool for menus that exploits the SDP [17] model, and a model of expected item groupings (i.e., FSM). Similarly to MenuOptimizer, we adapt SDP to predict the selection time of a technology from a grid layout, and we involve the user in the optimization process: by defining the trigger, users interactively provide the optimizer with fundamental information to produce the layout for defining the action.

III. OPTIMIZING IF-THEN RULES COMPOSITION

To clearly define and exemplify our approach, we introduce its implementation (*EUDoptimizer*, described in Section VI) in a scenario of trigger-action programming.

John owns many devices always connected to the Internet, and he is subscribed to various social networks and cloud services. John has just started to use EUDoptimizer to define trigger-action rules for customizing the joint behavior of his devices and services. John would like to have all the photos taken with his two smartphones saved in different places, e.g., for backup purposes, and for making them available on all his other devices. Furthermore, he would like to automatically share the photos on image sharing platforms. John opens EUDoptimizer and starts to define a trigger-action rule.

A. TRIGGER DEFINITION

For the definition of the trigger, EUDoptimizer shows John a grid layout in which the optimizer has found a trade-off for a) displaying technologies according to their usage probability, and b) grouping together technologies to maintain logical groups of functional-related devices or services. Thanks to the FSM model, which characterize devices and services on the basis of what they can do, photo and video technologies with similar functionality are grouped together (as shown later in Figure 3a), thus facilitating John in finding what he needs. Furthermore, thanks to SDP and since most people extensively use such technologies, they are displayed on the top of the layout. John select the iOS Photo technology, that allows him to define a trigger to monitor every time he takes a photo on his iOS smartphone.

B. ACTION DEFINITION

The definition of the action strongly depends on the definition of the trigger, i.e., the grid menu for defining the action should interactively change according to the chosen trigger. As for the trigger definition, EUDoptimizer previously calculated an optimized components layout to be used as an initial solution by considering usage probabilities and functional similarities. As soon as John selects iOS Photo

for defining the trigger, EUDoptimizer starts to refine the initial solution according to on the user choice. The idea is to find the best trade-off for promoting the elements most commonly associated with the selected trigger technology, without affecting groups of functionally similar technologies. When John wants to define the action, EUDoptimizer shows the grid layout of Figure 4. Not surprisingly, the functionality in which John is interested are not uncommon. At the top of the layout, John can find different technologies that allow him to reach his goals. With Dropbox, Google Drive, etc., John can save the photos taken with his smartphone on the cloud, thus making them available for all his other Internet-enabled devices. John decide to the define an action for saving the photos on his Google Drive folder. John is very satisfied of EUDoptimizer: now he can define many other rules to customize his other smartphone, an Android-based device, and to save and share his photos with many different online services.

IV. PREDICTIVE MODELS FOR TRIGGER-ACTION PROGRAMMING

The goal of our approach is to employ model-based optimization methods to *dynamically* redesign grid layouts in EUD interfaces in an *interactive* way, i.e., by considering the choices made by end users. Model-based optimization methods need to be supported by valid and comprehensive predictive models. One of the most recent model of menu performance is SDP [17]. We adapt such a model to work with grid layouts in EUD interfaces, which display technologies for defining triggers and actions. Furthermore, similarly to the menu optimizer proposed by Bailly *et al.* [14], we use SDP in combination with a model that takes into account the *expectation* of item groups, i.e., the expectations of users in finding certain items together. Such a novel model for item grouping, named Functionality Similarity Model (FSM), considers similarities between technologies in terms of the functionality they allow to define through their triggers and actions. Roughly speaking, a *Philips Hue* lamp shares some functionality with a *Hunter Duglas* blind for defining an action, because they both allow the increase or decrease of the brightness of a room. The *Android Location* service and the *Nest* surveillance camera, instead, allow the definition of triggers with the same final goal, i.e., to monitor when someone is entering a place. To discover such similarities, we use the Semantic Web framework, and, in particular, the EUPont ontology.[4]

A. SDP: SEARCH DECISION POINTING

SDP is a state-of-the-art model of human performance in linear menu search. It incorporates both Hick-Hyman and Fitts' laws, and integrates a transition from novice to expert performance. We adapt the model to be used for grid layouts, i.e., a particular type of menu, by using the euclidean distance between items. SDP predicts the selection time of an item i in a menu with the following formula [17]:

$$T_i = (1 - e_i) \cdot T_{st} + e_i \cdot T_{dt} + T_{pt}, \quad (1)$$

where:

- e_i models the user expertise with the item i.
- T_{st} is the **search time**, i.e., the time to localize the item, linear with the total number of items when the user is inexperienced.
- T_{dt} is the **decision time**, i.e., the time to decide from among items, given by the Hick-Hyman law once the user becomes expert with the item i.
- T_{pt} is the **pointing time**, i.e., the time to "point" the item, described according to the Fitts' law, which predicts that items closer to the top are faster to select.

To predict the average performance of an entire menu, SDP uses the following formula:

$$T_{avg} = \sum_{i=1}^{n} p_i \cdot T_i, \quad (2)$$

where p_i is the probability of item i being selected, i.e., its usage probability, and n is the number of menu items.

The probability function p_i can be reformulated as the *frequency probability*, i.e., the probability of the technology i to be used in a trigger or in an action. The idea is to move towards the top of the grid layout the technologies most commonly selected as triggers or actions. When the user select a technology i for the trigger, the *SDP* model interactively changes in $SDP(i)$, where p is reformulated as the *bigram probability* p_{ij} of selecting an action technology j after having selected the technology i for the trigger. The idea is to move towards the top of the grid layout the action technologies most commonly associated with the trigger technology i.

The parameters we used for the SDP model are the same used both in the original paper [17] and in the optimizations carried out in MenuOptimizer [14].

B. FSM: FUNCTIONALITY SIMILARITY MODEL

We propose and define the Functionality Similarity Model to allow the optimizer to produce groups of technologies that are *functionally* correlated, even if they are heterogeneous technologies. The model exploits the EUPont [8] ontology, a semantic representation for trigger-action programming that offers a three-layer hierarchical categorization of triggers and actions in terms of their final functionality. The three categories, named Low-Level, Medium-Level, and High-Level, represent three different levels of abstraction in which triggers and actions can be expressed and categorized. For example, the Low-Level category *"turn lamp x on"* includes all the actions aiming at turning a specific lamp (identified by x) on. Such a category can be further abstracted in *"turn the lights on"* (Medium-Level category) and *"illuminate a place"* (High-Level category), respectively. With this hierarchical representation, the *"illuminate a place"* category includes all the actions for turning lights on, along with other actions that allow to reach the final goal of illuminating a place, e.g., opening a blind.

[4] https://elite.polito.it/ontologies/eupont.owl last visited on April 10, 2018

By considering this EUPont categorization, the Trigger Functionality Association (FA_t) between two technologies i and j is calculated as:

$$FA_t(i,j) = \alpha_f \cdot LL_t(i,j) + \beta_f \cdot ML_t(i,j) + \gamma_f \cdot HL_t(i,j), \quad (3)$$

where:

- $LL_t(i,j)$ is the number of Low-Level categories for which i and j both offer at least one trigger.
- $ML_t(i,j)$ is the number of Medium-Level categories for which i and j both offer at least one trigger.
- $HL_t(i,j)$ is the number of High-Level categories for which i and j both offer at least one trigger.
- α_f, β_f, and γ_f sum to 1, and are used to weights the three elements modeled in FSM, i.e., LL_t, ML_t, and HL_t.

In the same way, the Action Functionality Association (FA_a) between i and j is calculated as:

$$FA_a(i,j) = \alpha_f \cdot LL_a(i,j) + \beta_f \cdot ML_a(i,j) + \gamma_f \cdot HL_a(i,j) \quad (4)$$

In our implementation, we empirically set $\alpha_f = 0.6$, $\beta_f = 0.3$, and $\gamma_f = 0.1$. This means that technologies that shares Low-Level (very specific) categories are more similar than technologies that only shares Medium or High-Level categories, that are intrinsically more abstract.

To use the model in a minimization problem, we exploit the pairwise Functionality Associations to compute the Functionality Incoherence score of a given grid menu, both for the definition of triggers and actions (FI_t and FI_a):

$$FI_t = \sum_{i=1}^{n} \sum_{j=i+1}^{n} FA_t(i,j) \cdot d(i,j), \quad (5)$$

$$FI_a = \sum_{i=1}^{n} \sum_{j=i+1}^{n} FA_a(i,j) \cdot d(i,j), \quad (6)$$

where $d(i,j)$ is the euclidean distance between objects i and j in the grid layout.

As shown in Figure 3b, the FSM model has desirable effects on the optimizer: technologies that offer triggers (or actions) with similar functionality tend to be pulled together, while unrelated technologies are moved away.

V. OPTIMIZATION PROBLEM AND METHODS

In this section, we first formulate the optimization problem by defining the objective function we used to explore the design space. Then, we present the optimization methods we used to attack the problem, based on metaheuristic strategies.

A. PROBLEM FORMULATION

To explore the design space looking for "good" or "desirable" grid menu alternatives, we define a multi-objective task. The goal of the optimizer is to minimize a weighted combination of the outputs of the two models M_i exploited by *EUDoptimizer*, i.e., SDP and FSM:

$$\min \sum_{i=1}^{2} \lambda_i \cdot M_i, \quad (7)$$

where the sum of all the weights λ_i is 1. In our implementation, we empirically set the λ values thanks to the trials performed in the technical assessment of the implemented algorithms (Section VII).

To make the single objectives less sensitive to weight selection, we normalized each M_i with the objective value θ_i calculated for an initial point x_0:

$$\theta_i = M_i(x_0). \quad (8)$$

The problem for designing the grid menu for trigger definition is therefore:

$$\min (\lambda_1 \cdot SDP + \lambda_2 \cdot FI_t) \quad (9)$$

while for the action the problem is:

$$\min (\lambda_1 \cdot SDP + \lambda_2 \cdot FI_a) \quad (10)$$

Based on the interaction of the user, i.e., when the user select a technology i, the problem for the action changes, and becomes:

$$\min (\lambda_1 \cdot SDP_i + \lambda_2 \cdot FI_a) \quad (11)$$

B. SOLVING THE OPTIMIZATION PROBLEM

The problem of designing menu systems, both linear and grid-based, can be formulated as a Quadratic Assignment Problem (QAP) [14]. Developed in operational research, QAP [18] allows the modeling of relationships between elements of two sets to minimize the total pairwise cost. In designing menus, m items have to be assigned to m predetermined locations in order to maximize usability, e.g., expected selection time, menu coherence, etc. QAP is a NP-hard problem, and it is considered one of the hardest optimization problems since general instances of size $m > 20$ cannot be solved to optimality. In our case, for example, m technologies (typically with $m > 200$) can be organized in $m!$ ways. Given the complexity of the problem, we cannot claim global optimality, e.g., through exact methods. To attack the problem, we exploit metaheuristic strategies. A heuristic is a technique that seeks near-optimal solutions at a reasonable computational cost without guaranteeing optimality. Some heuristic methods, however, can get easily trapped in a local optimum: metaheuristics methods modify their use of heuristics methods as optimization progresses [19]. Our implementation, described in the next section, supports two metaheuristics successfully used for QAP problems, i.e., Simulated Annealing [20], [21] and Ant Colony System [22]. Simulated Annealing is based on mimicking the metal annealing processing and exploits local and random search in a exploration/exploitation scheme. The main advantage of simulated annealing is its ability to avoid being trapped in local optima. In fact, a neighboring solution is not considered only when it yields to a better objective value: with a certain probability the solution is accepted even if it does not improve the objective. The pseudo-code for the simulated annealing is reported in Algorithm 1.

Instead, Ant Colony System is based on the biological metaphor of an ant colony foraging for food, in which multiple searchers cooperate to produce solutions according to a memory of past solutions. The pseudo-code of ACS is presented in Algorithm 2.

Algorithm 1 Simulated Annealing

Let $s = s_0$
for $k = 0$ through k_{max} **do**
  $T \leftarrow$ temperature(k/k_{max})
  Pick a random neighbour, $s_{new} \leftarrow$ neighbour(s)
  if $P(E(s), E(s_{new}), T) \geq$ random$(0, 1)$ **then**
    $s \leftarrow s_{new}$
Output: the final state s

Algorithm 2 Ant Colony System

while (not converged) **do**
  Position each ant in a starting node
  repeat
    for all ant **do**
      Chose next node with the transition rule
      Apply local pheromone update
  until every ant has built a solution
  Update best solution
  Apply global pheromone update

VI. EUDOPTIMIZER

In this section, we describe *EUDoptimizer*, an implementation of our approach on top of IFTTT. Despite our approach is generic, i.e., it can be applied to any grid-based EUD interface for trigger-action programming, we chose IFTTT due to the popularity of the platform and the availability of real usage data [7]. To maintain a high level of interactivity, we implemented a client-server architecture between the optimizer and the user interface for composing trigger-action rules.

A. DATA AND MODELS

We exploited a large dataset of trigger-action rules obtained by Ur *et al.* [7] with a web scrape of the public rules shared on IFTTT as of September 2016. The dataset is composed of 295,156 rules created by 129,206 different authors, and it includes more than 200 different technologies, ranging from smart home devices to web applications. Besides the information about triggers and actions, each rule also includes a sharing information, i.e, the number of times that it has been reused by other users, thus providing a direct measure of its popularity. We used the sharing information of each rule, normalized between 0 and 1, to calculate the *frequency probabilities* and the *bigram probabilities*, to be used in SDP model. Furthermore, we linked each technology with the corresponding EUPont entity by using the translation process described in [6], and we calculated the Functionality Associations (FA) to be used in the FSM model.

B. USER INTERFACE

By exploiting the information extracted from the dataset, we modeled a user interface after IFTTT with the AngularJS framework.[5] The interface, shown in Figure 2, allows the composition of trigger-action rules exactly as in IFTTT. For defining a trigger, for example, users have to click on the "this" button (Figure 2a), and then select the desired technology from a grid layout (Figure 2b). Finally, they can select the specific trigger to be monitored (Figure 2c), by filling any required details. To compare *EUDoptimizer* with the original IFTTT, we realized two versions of the same interface, namely *IFTTT* and *EUDoptimizer*. The difference is obviously in the grid menu of Figure 2b: while for *EUDoptimizer* the layout of such a menu is provided by the optimizer, in *IFTTT* it reflects the same menu available on the original platform.

C. OPTIMIZER

We implemented two different solvers in Python, based on Simulated Annealing (SA) and Ant Colony System (ACS), respectively. We executed them on a regular laptop (a 2015 MacBook Pro with a 2.7 Ghz Intel Core i5 and 8 GB of RAM), separately. To define the algorithm parameters, we empirically run a set of 100 optimizations by varying the parameter values. Both optimizers provide the same functionality. They initially generates two grid layouts T1 and A1 (for defining triggers and actions, respectively) by using the two "static" versions of the optimization problem (Equation 9 and Equation 10). Such layouts are periodically recalculated to reflect changes in the probability distributions, e.g., due to new rules defined by the user. As soon as the user selects a

[5] `https://angularjs.org/` last visited on February 12, 2018

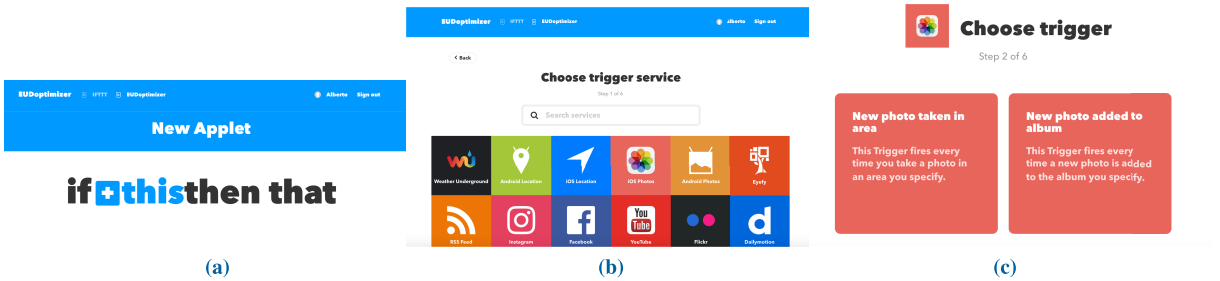


FIGURE 2. Some screenshots of the user interface used in the empirical evaluation. The interface resembles IFTTT and allows the composition of trigger-action rules with both the IFTTT version and the *EUDoptimizer* enhanced version. In the IFTTT terminology, rules are named applets, while technologies are named services. (a) New rule page. (b) Trigger technology selection step. (c) Trigger selection step.

TABLE 1. Results of the simulated annealing for the trigger layout with 100, 1,000, 5,000, and 10,000 iterations.

Iterations	Time [sec]	Obj. value
100	12	0.99
1,000	106	0.98
5,000	536	0.47
10,000	1,031	0.43

TABLE 2. Results of the ant colony system for the trigger layout with 100, 1,000, 5,000 and 10,000 iterations.

Iterations	Time [sec]	Obj. value
100	178	0.89
1,000	1,896	0.88
5,000	10,012	0.86
10,000	20,515	0.82

technology for defining the trigger, each optimizer interactively receive the information. Starting from the layout A1, each of them starts to explore the problem described by the Equation 11 to refine the initial layout to take into account the user's choice. When the user finishes the trigger definition, i.e., she completes it with all the required parameters, each optimizer generates a grid menu A2, an improved version of A1 in which the technologies that are most likely to be used with the selected trigger are promoted towards the top.

VII. TECHNICAL ASSESSMENT

To assess the feasibility of our approach, and to evaluate which optimizer provide better solutions, we executed the SA and the ACS solvers *"off-line,"* by changing the number of iterations of the algorithms. During the trials, we also tuned the parameters of the algorithms, along with the weights of our optimization problem.

A. TRIGGER DEFINITION

We first executed the optimizers for calculating a grid layout for trigger definition, thus solving Equation 9. Table 1 and Table 2 report the results obtained with 100, 1,000, 5,000, and 10,000 iterations with SA and ACS, respectively. Despite the ACS solver provides better solutions for 100 and 1,000 iterations, the SA solver performs better with a higher number of iterations. Furthermore, SA is faster than ACS in all cases. Figure 3 shows two screenshots of the grid menu calculated with 10,000 iterations of SA, obtained in less than 20 minutes.

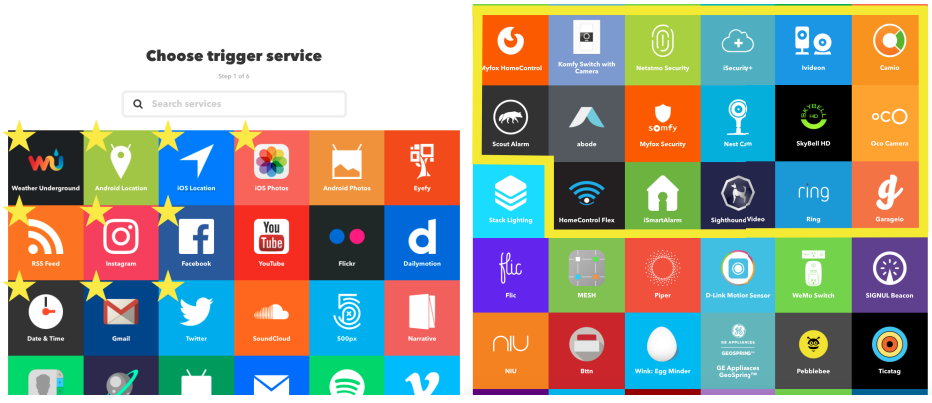
We can observe that:

- The 10 most frequently used technologies[6] for defining triggers are prominently placed in the 10 positions closer to the top of the grid menu (Figure 3a). Their *functional* similarities are pulled together and organized in logical groups, e.g., locations (*Android Location*, *iOS Location*), photos and videos (*iOSPhoto*, *Android Photo*, *Eyefy*, *Youtube*, *Flickr*, *Dailymotion*, and *500px*), etc.
- Figure 3b further highlights that technologies with *functional* similarities are pulled together in logical groups. The figure, in particular, shows a huge group of hubs, cameras, and doorbells, all related to home security.

B. ACTION DEFINITION

We then executed the optimizers for calculating the action grid menu. In this case, we initially used the solvers to calculate (with 10,000 iterations) an initial solution for the menu, i.e., by solving Equation 10. Then, we manually fixed the selected trigger technology to *iOS Photo*, one of the most used trigger in the data set, and we tested the optimizers to solve the problem defined by Equation 11, i.e., the interactive optimization. Table 3 and Table 4 report the optimization results for SA and ACS, respectively. Also in this case, SA performed better than ACS with 10,000 iterations. Figure 4 shows the top part of the best grid menu obtained with 10,000 iterations by SA. Both Figure and Table confirm that the SA solver produces good solutions in a reasonable amount of time. In fact, 9 of the 10 technologies most frequently associated with *iOS Photo* are presented on the top

[6]according to the dataset of Ur *et al.* [7]



(a) (b)

FIGURE 3. Optimized grid layout for defining triggers calculated with the Simulated Annealing solver (10,000 iterations). (a) shows the top of the layout (the added stars indicate the 10 most frequently used technologies). (b) shows the bottom part; a yellow border highlights a logical group of technologies related to home security.

TABLE 3. Results of the simulated annealing for the action layout when the selected trigger technology is *iOS Photo*.

Iterations	Time [sec]	Obj. value
100	10	1.01
1,000	91	0.98
5,000	468	0.47
10,000	895	0.45

TABLE 4. Results of the ant colony system for the action layout when the selected trigger technology is *iOS Photo*.

Iterations	Time [sec]	Obj. value
100	171	0.88
1,000	1669	0.83
5,000	11547	0.82
1,0000	20515	0.82

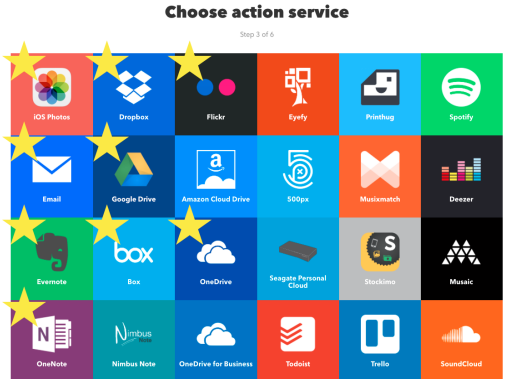


FIGURE 4. Optimized grid layout for actions calculated with SA when the selected trigger technology is *iOS Photo* (10,000 iterations). Stars indicate the most frequently used technologies: 9 out of 10 are at the top of the grid layout.

of the grid layout. Furthermore, there are logical groups of related technologies that allow the definition of similar functionality, e.g., *One Note*, *Nimbus Note*, and *Evernote*.

VIII. USER STUDY

Thanks to the technical assessment, we selected the SA solver to be used in *EUDoptimizer*, and we compared the optimized interface with *IFTTT* in a user study with 12 participants, by asking participants to compose trigger-action rules with both interfaces. The goal was to understand whether *EUDoptimizer* a) improved the user performance, i.e., the time needed for composing trigger-action rules, and b) reduced the cognitive load in the composition of trigger-action rules with respect to the ''normal'' version. In this section, we describe the study, and we report the quantitative and qualitative results used for investigating the time variable and the cognitive effort, respectively.

A. STUDY DESCRIPTION

1) DESIGN AND PROCEDURE

We devised a within-subject user study, where we considered the interface version (*IFTTT* vs. *EUDoptimizer*) as the independent variable. We first submitted participants an initial questionnaire to collect demographic data and information about their programming skills and their previous experience with IFTTT. Then, we introduced them to trigger-action programming and to the IFTTT environment, and we explained the nature of the study. During this phase, we showed the interface to the users, by composing a trigger-action rule in *IFTTT* as an example. After the training phase, we asked participants to complete 6 similar tasks related to the composition of trigger-action rules with both interface versions, without telling them which version they were using. Interface versions and tasks were fully counterbalanced between the participants. The study was carried out in a university office, and took about 30 minutes per participant. All the sessions were audio recorded.

2) PARTICIPANTS

We recruited 12 participants (4 female and 8 male) with a mean age of 25.91 years ($SD = 4.48$, $range : 19 - 34$). To consider users with and without programming skills, participants were recruited from different background, i.e., Education, Biology, Aerospace Engineering, Management Engineering, and Computer Engineering. 3 participants were undergraduate students, 7 were Ph.D. students, while 2 where post-doc researchers, all coming from different universities. On a Likert scale from 1 (No experience at all) to 5 (I am an expert), participants declared their programming experience level ($M = 3$, $SD = 1.04$), and their experience with IFTTT ($M = 1.67$, $SD = 0.89$).

3) TASKS

In the study, each task consisted in the composition of a single trigger-action rule. We defined 6 different tasks that asked participants to replicate trigger-action rules previously extracted from the IFTTT dataset [7]. To explore the full range of possible alternatives, e.g., to evaluate *EUDoptimizer* both with commonly and rarely used technologies, we first divided the dataset in three layers by grouping together the most common rules (i.e., shared more than 10,000 times), the common rules (i.e., shared 1,000 to 10,000 times), and the uncommon ones (i.e., shared fewer than 1,000 times). Then, we randomly selected 2 rules for each category. The rules, rephrased for the sake of readability, were:

Most common rules

- IF the *Weather Underground* service notifies that tomorrow's forecast call for rain, THEN use *Notifications* to send me a notification.
- IF I share a photo on *Instagram*, THEN add the file on my *Dropbox*.

Common rules

- IF I add a photo on *iOS Photo*, THEN add the file on my *Google Drive*.
- IF I receive a new labeled email on *Gmail*, THEN create a note on *Evernote*.

Uncommon rules

- IF the laundry cycle of my *Samsung Washer* is finished, THEN add an event on *Google Calendar*.

- IF the *Nest Cam* recognizes a new sound or motion event, THEN turn the *Philips Hue* on.

4) MEASURES

For each task completion (with both the *IFTTT* and the *EUDoptimizer* interface), we measured the following times:

- **Trigger Time (TT).** The time for selecting the technology to define the trigger from the grid layout.
- **Action Time (AT).** The time for selecting the technology to define the action from the grid layout.
- **Rule Time (RT).** The time for composing the entire rule, composed of the Trigger and Action times (TT and AT), and the time needed for completing the specific trigger and action, e.g., for filling the required details.

Furthermore, we extracted any consideration made by the participants from the audio registrations.

TABLE 5. Average trigger time (TT), action time (AT), and rule time (RT) with the *IFTTT* interface.

Measure	Mean [sec]	Std. Error
TT	37.29	13.44
AT	20.87	11.18
RT	81.77	23.14

TABLE 6. Average trigger time (TT), action time (AT), and rule time (RT) with the *EUDoptimizer* interface.

Measure	Mean [sec]	Std. Error
TT	21.96	11.02
AT	8.26	8.97
RT	51.68	18.28

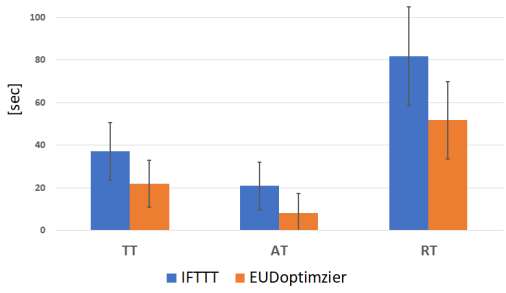


FIGURE 5. Average trigger time (TT), action time (AT), and rule time (RT) compared between the *IFTTT*-like interface and its *EUDoptmizer* enhanced version. All the time measures are lower in the optimized interface.

B. QUANTITATIVE RESULTS

Table 5, Table 6, and Figure 5 show and compare the measures obtained with the *IFTTT* and the *EUDoptimizer* interfaces, respectively, averaged for all participants and tasks.

All the time measures were lower with the optimized interface. Therefore, *EUDoptimizer* improved the selection time in the trigger and in the action definition. Such improvements were reflected in the time needed by end users to compose a trigger-action rule (RT): the *EUDoptimizer* interface, in fact, allowed participants to define rules faster that the *IFTTT* interface.

To investigate whether the differences in the measures were significant, we analyzed the effect of the independent variable (*IFTTT* vs. *EUDoptimizer*) over the three dependent variables (TT, AT, and RT) with a one-way repeated measures ANOVA carried out in SPSS. The Mauchly's sphericity test was satisfied in all the three analysis. There was a significant main effect of the used interface on TT ($F(1, 11) = 8.30, p < .05$), AT ($F(1, 11) = 12.46, p < .05$) and RT ($F(1, 11) = 15.82, p < .05$). For all the 3 dependent variables, a post-hoc test with the Bonferroni correction revealed that the mean differences between the two interfaces were statistically significant ($p < .05$), thus confirming the evidence that *EUDoptimizer* improved the selection time in the trigger and action definition phases, and the overall composition time of trigger-action rules.

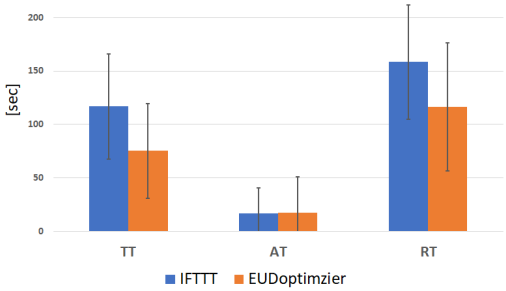


FIGURE 6. Average trigger time (TT), action time (AT), and rule time (RT) compared between the *IFTTT*-like interface and its *EUDoptmizer* version for the two uncommon rules.

We can say that *EUDoptimizer* improves the composition of trigger-action rules by reducing the time effort in the definition of triggers and actions. To further demonstrate such a statement, we separately analyzed the measures for the two *uncommon rules*, only (Figure 6). We were particularly interested in evaluating the potential of *EUDoptimizer* in the worst case. In fact, since we used the dataset to weight items by their frequency, technologies involved in the uncommon rules were not placed in the first position of the grid layouts.

We found that the optimized interface improved the definition of triggers also for the two uncommon rules. The TT measure was lower with the *EUDoptimizer* interface (75.32 ± 44.25 sec *vs.* 116.81 ± 49.19 sec). On average, the selection of a technology for defining actions was instead performed with similar performances by participants (16.67 ± 24.21 sec *vs.* 17.70 ± 33.43 sec). However, the time for composing the entire rules (RT measure) was considerably lower with the *EUDoptimizer* interface (116.52 ± 59.82 sec *vs.* 158.31 ± 53.17 sec).

C. QUALITATIVE RESULTS

Qualitative data extracted from the audio recorded files show that the benefit of *EUDoptimizer* are not restricted to time performance, only. The indications of the participants, in fact,

suggest that *EUDoptimizer* reduces the cognitive effort to find technologies by ordering the components layout with logical groups of *functional-related* elements. In particular, we found that the majority users were frustrated in using the *IFTTT* interface. Without knowing the differences between the two evaluated interface versions, a participant using the *IFTTT* interface said *"I hate this task, it's very difficult to find the desired technology, I have already looked over the menu 4 times!"* Other 4 participants pointed out that technologies seemed to be displayed in a random order, thus making impossible to apply any search criterion. One participant said *"I am forced to search the technology by looking sequentially to all the listed elements, from the top to the end of the grid."* On the contrary, *EUDoptimizer* provided more support for selecting the desired technologies. 5 participants were happy of the "logical" groups of technologies showed by *EUDoptimizer*. A participant, for example, said: *"in this interface elements are ordered meaningfully. This helps me to find what I need."* Another participant said *"here the Samsung Washer is near to other appliances of the same type, it's easy to find it."* The other participants were instead happy because the technologies they needed, especially for defining actions, were displayed towards the top of the grid layout. A participant said: *"I like this interface [EUDoptimizer] because it proposes me the technologies I need in the first positions and I can immediately select them."*

IX. DISCUSSION

Trigger-action programming allows end users to customize their smart devices and web-based services on the basis of their personal needs, but becomes more and more challenging as the number of available technologies and smart environments increases. We proposed and successfully adopted an approach to integrate optimization methods in contemporary EUD interfaces. The *EUDoptimizer* implementation, in particular, suggests that the approach is valuable. *Off-line* results obtained with 10,000 iterations ($\sim$ 15/20 minutes on a regular laptop) are promising, and show that satisfactory solutions can be obtained in a reasonable amount of time. This is confirmed by the results of the user study, where *EUDoptimizer* performed the optimizations in *real-time*, by interacting with the participants. By comparing *EUDoptimizer* with *IFTTT*, in particular, we found that trigger-action rules were composed in less time with the optimized interface. Even for the most uncommon rules, for which technologies were not placed on the top of the layouts, *EUDoptimizer* partially reduced the time effort needed by participants to complete the tasks. Furthermore, qualitative data extracted from the user study suggest that *EUDoptimizer* reduces the cognitive effort to compose IF-THEN rules by redesigning the grid layouts of EUD interfaces with a focus on the final *functionality* of the supported technologies.

Despite these advantages should be further studied, integrating optimization methods in EUD interfaces, as in *EUDoptimizer*, could help end users better deal with trigger-action programming. With the continuous spread of new smart objects and online services, in fact, an optimized EUD interface could reduce time and cognitive efforts needed by end users for selecting elements between hundreds of supported technologies, and could open up new possibilities for users to program their devices and services.

X. CONCLUSIONS

In this paper we investigated a new approach to interactively assist end users in composing IF-THEN rules with an optimizer in the loop. The goal was to *dynamically* redesign layouts in EUD interfaces in an *interactive* way, i.e., by considering the choices made by users. To reach our goal, we adapted a state-of-the-art predictive model of user performance in menu search, and we define a novel model to organize technologies (i.e., various smart devices, online services, ...) on the basis of their final functionality. We used different optimization algorithms to explore the design space, and we integrated the optimization methods in *EUDoptimizer*, an implementation of our approach on top of IFTTT. The implementation showed that the optimization can produce satisfactory solutions in a reasonable amount of time. Moreover, a user study with 12 participants confirmed that *EUDoptmizer* reduces the time and cognitive efforts needed by end users to compose trigger-action rules. There are several opportunities to improve the approach. The models, for example, could be further analyzed: are there any other criteria, beside the functionality similarity, to refine the optimization approach? Which criteria has a greater impact on end user efforts? Such considerations are guiding our current works.

REFERENCES

[1] G. Desolda, C. Ardito, and M. Matera, "Empowering end users to customize their smart environments: Model, composition paradigms, and domain-specific tools," *ACM Trans. Comput.-Hum. Interact.*, vol. 24, no. 2, Apr. 2017, Art. no. 12.

[2] H. Lieberman, F. Paternò, M. Klann, and V. Wulf, "End-user development: An emerging paradigm," in *End User Development*. Dordrecht, Netherlands: Springer, 2006, pp. 1–8.

[3] D. Evans, "The Internet of Things: How the next evolution of the Internet is changing everything," Cisco Internet Bus. Solutions Group, San Jose, CA, USA, White Paper, Apr. 2011. [Online]. Available: https://www.cisco.com/c/dam/en_us/about/ac79/docs/innov/IoT_IBSG_0411FINAL.pdf

[4] T.-H. K. Huang, A. Azaria, and J. P. Bigham, "InstructableCrowd: Creating IF-THEN rules via conversations with the crowd," in *Proc. CHI Conf. Extended Abstr. Hum. Factors Comput. Syst.*, New York, NY, USA, May 2016, pp. 1555–1562.

[5] B. R. Barricelli and S. Valtolina, "Designing for end-user development in the Internet of Things," in *Proc. 5th Int. Symp. End-User Develop. (IS-EUD)*. Cham, Germany: Springer, May 2015, pp. 9–24.

[6] F. Corno, L. De Russis, and A. M. Roffarello, "A semantic Web approach to simplifying trigger-action programming in the IoT," *Computer*, vol. 50, no. 11, pp. 18–24, Nov. 2017.

[7] B. Ur *et al.*, "Trigger-action programming in the wild: An analysis of 200,000 IFTTT recipes," in *Proc. CHI Conf. Hum. Factors Comput. Syst.*, New York, NY, USA, May 2016, pp. 3227–3231.

[8] F. Corno, L. De Russis, and A. M. Roffarello, "A high-level approach towards end user development in the IoT," in *Proc. CHI Conf. Extended Abstr. Hum. Factors Comput. Syst.*, New York, NY, USA, May 2017, pp. 1546–1552.

[9] A. Oulasvirta, "User interface design with combinatorial optimization," *Computer*, vol. 50, no. 1, pp. 40–47, Jan. 2017.

[10] S. Zhai, M. Hunter, and B. A. Smith, "The metropolis keyboard—An exploration of quantitative techniques for virtual keyboard design," in *Proc. 13th Annu. ACM Symp. Interface Softw. Technol.*, New York, NY, USA, Nov. 2000, pp. 119–128.
[11] A. Oulasvirta *et al.*, "Improving two-thumb text entry on touchscreen devices," in *Proc. SIGCHI Conf. Hum. Factors Comput. Syst.*, New York, NY, USA, Apr. 2013, pp. 2765–2774.
[12] K. Z. Gajos, J. O. Wobbrock, and D. S. Weld, "Improving the performance of motor-impaired users with automatically-generated, ability-based interfaces," in *Proc. SIGCHI Conf. Hum. Factors Comput. Syst.*, New York, NY, USA, Apr. 2008, pp. 1257–1266.
[13] S. Matsui and S. Yamada, "Genetic algorithm can optimize hierarchical menus," in *Proc. SIGCHI Conf. Hum. Factors Comput. Syst.*, New York, NY, USA, Apr. 2008, pp. 1385–1388.
[14] G. Bailly, A. Oulasvirta, T. Kötzing, and S. Hoppe, "MenuOptimizer: Interactive optimization of menu systems," in *Proc. 26th Annu. ACM Symp. Interface Softw. Technol.*, New York, NY, USA, Oct. 2013, pp. 331–342.
[15] K. Todi, D. Weir, and A. Oulasvirta, "Sketchplore: Sketch and explore with a layout optimiser," in *Proc. ACM Conf. Designing Interact. Syst.*. New York, NY, USA, Jun. 2016, pp. 543–555.
[16] P. Salem, "User interface optimization using genetic programming with an application to landing pages," in *Proc. ACM Hum.-Comput. Interact.*, vol. 1, Jun. 2017, Art. no. 13.
[17] A. Cockburn, C. Gutwin, and S. Greenberg, "A predictive model of menu performance," in *Proc. SIGCHI Conf. Hum. Factors Comput. Syst.*, New York, NY, USA, Apr. 2007, pp. 627–636.
[18] S. S. Rao, *Engineering Optimization: Theory and Practice*, 4th ed. Hoboken, NJ, USA: Wiley, 2009.
[19] F. Glover and M. Laguna, *Tabu Search*. Norwell, MA, USA: Kluwer, 1997.
[20] M. R. Wilhelm and T. L. Ward, "Solving quadratic assignment problems by 'simulated annealing,'" *IIE Trans.*, vol. 19, no. 1, pp. 107–119, Jul. 1987.
[21] D. T. Connolly, "An improved annealing scheme for the QAP," *Eur. J. Oper. Res.*, vol. 46, no. 1, pp. 93–100, May 1990.
[22] M. Dorigo and L. M. Gambardella, "Ant colony system: A cooperative learning approach to the traveling salesman problem," *IEEE Trans. Evol. Comput.*, vol. 1, no. 1, pp. 53–66, Apr. 1997.



FULVIO CORNO has been an Associate Professor with the Department of Control and Computer Engineering, Politecnico di Torino, since 2002, where he is currently the Leader of the e-Lite Research Group and focuses on ambient intelligence systems by integrating novel interaction modalities with the IoT architectures. He is a member of the IEEE, the IEEE Computer Society, and the ACM.



LUIGI DE RUSSIS has been an Assistant Professor with the Department of Computer and Control Engineering, Politecnico di Torino, since 2018. His current research interest includes human–computer interaction, with a particular interest on interaction techniques applied to complex settings (such as ambient intelligence systems). He is a member of the IEEE, the IEEE-HKN, the IEEE Computer Society, and the ACM.



ALBERTO MONGE ROFFARELLO is currently pursuing the Ph.D. degree with the Department of Computer and Control Engineering, Politecnico di Torino, advised by F. Corno. His current research interests include Semantic Web and human–computer interaction, with a particular interest on end user customization of ambient intelligence systems. He is a Student Member of the IEEE and the ACM. He is also an IEEE-HKN Member.