Learning Alternative Ways of Performing a Task

David Nieves
Valencian Research Institute for Artificial Intelligence (VRAIN), Universitat Politècnica de València, Spain
daniecor@dsic.upv.es

María José Ramírez-Quintana
Valencian Research Institute for Artificial Intelligence (VRAIN), Universitat Politècnica de València, Spain
mramirez@dsic.upv.es

Carlos Monserrat (corresponding author)
Valencian Research Institute for Artificial Intelligence (VRAIN), Universitat Politècnica de València, Spain
cmonserr@dsic.upv.es

César Ferri
Valencian Research Institute for Artificial Intelligence (VRAIN), Universitat Politècnica de València, Spain
cferri@dsic.upv.es

José Hernández-Orallo
Valencian Research Institute for Artificial Intelligence (VRAIN), Universitat Politècnica de València, Spain
jorallo@dsic.upv.es

Learning Alternative Ways of Performing a Task

D. Nieves, MJ. Ramírez-Quintana, C. Monserrat[1], C. Ferri, J. Hernández-Orallo

Valencian Research Institute for Artificial Intelligence (VRAIN), Universitat Politècnica de València, Spain

Abstract

A common way of learning to perform a task is to observe how it is carried out by experts. However, it is well known that for most tasks there is no unique way to perform them. This is especially noticeable the more complex the task is because factors such as the skill or the know-how of the expert may well affect the way she solves the task. In addition, learning from experts also suffers of having a small set of training examples generally coming from several experts (since experts are usually a limited and expensive resource), being all of them positive examples (i.e. examples that represent successful executions of the task). Traditional machine learning techniques are not useful in such scenarios, as they require extensive training data. Starting from very few executions of the task presented as activity sequences, we introduce a novel inductive approach for learning multiple models, with each one representing an alternative strategy of performing a task. By an iterative process based on generalisation and specialisation, we learn the underlying patterns that capture the different styles of performing a task exhibited by the examples. We illustrate our approach on two common activity recognition tasks: a surgical skills training task and a cooking domain. We evaluate the inferred models with respect to two metrics that measure how well the models represent the examples and capture the different forms of executing a task showed by the examples. We compare our results with the traditional process mining approach and show that a small set of meaningful examples is enough to obtain patterns that capture the different strategies that are followed to solve the tasks.

Keywords: task learning, inductive learning, process mining, identifying strategies

1. Introduction

Nowadays, humans learn the execution of a complex task through three steps: studying the description of the task, watching video executions of the task (usually performed by experts) or real-life demonstrations of it, and, finally, executing the task under expert supervision several times (Ericsson, 2009). This way of acquiring the skills needed to perform a task is expensive in time and resources. Besides, the lack of continuous supervision may induce mistakes because of the limited experience of the operators or lack of attention

[1]Corresponding author: cmonserr@dsic.upv.es

because of the repetitiveness of the task. In this sense, artificial intelligence (AI), and machine learning (ML) in particular, is making it possible to help people in their daily lives by learning models about their tasks. These models can then be integrated into direct assistance systems, such as task training learning environments, supervisory contexts in order to avoid human mistakes, or machine-human collaboration contexts, where the machine is used as an assistant for a complex task. Research in this direction can be found in fields such as *Ambient Intelligence* (Camacho et al., 2014), *Context-aware systems* (Hong et al., 2009), *Advanced Driving Assistants* (Škrjanc et al., 2018) or *Surveillance systems* (Kardas & Cicekli, 2017), to name a few.

However, traditional ML techniques require large volumes of data in order to infer models of common tasks. In many areas, such as surgery, the access to training examples is very costly, as these are very complex tasks that very few people know how to perform or require specific permissions or instrumental. In these cases, the knowledge acquisition has to be done from a small set of examples, with examples being a series of steps that an expert carried out to complete the task from beginning to end. Besides, this learning process must consider that any process can be performed in several ways; experts can present different styles when executing the task (in many cases, involuntarily) and can contain noise (interpreted as non-essential activities of the task).

Other approaches (Yang et al., 2017; Blum et al., 2008) can generalise a pattern of the task from a few positive examples. However, they only can learn one general pattern from the set of provided examples. The same case occurs with methods such as 'Fuzzy Mining' (Günther & van der Aalst, 2007) or the Alpha-algorithm (Alves de Medeiros et al., 2004). The previous solutions can lead to incorrect learning of the task and even dangerous learning. Let us illustrate the case with an example. Imagine a scenario where a system has to supervise the task of cooking a carrot soup. Table 1 shows the vocabulary of activities involved in this task. We have two examples of how to cook this dish (expressed as sequences of activities): "SABCDFGHIMOPT" and "SABCEFGJKLNOPT". These examples show two different forms of cooking: using a microwave or a kitchen stove. If we want to learn a general rule from these two examples, a possible generalisation is the expression "SABC {D|E}FG{HIM|JKLN}OPT", where "|" means a disjunction between two activity groups for the portion of the task delimited by curly brackets. The two given examples are covered by the model, but there are other valid sequences of activities according to this model that are not safe. For instance, the sequence "SABCEFGHIMOPT", where the person takes a metal pot, put the ingredients in, and, then, put the metal pot into the microwave to prepare the soup. We need a learning procedure and a representation language that is able to find the right trade-off between an overgeneralisation covering almost everything and an overspecialisation that is simply the sequence composed by the disjunction of the two examples "{SABCDFGHIMOPT|SABCEFGJKLNOPT}". Additionally, there can be examples that contain noise, for instance the sequence "SABBCDFGHIMOPT" is wrong, as the person washes the carrot twice (activity B) before cutting it. To avoid an overgeneralisation, given that we do not have negative examples, the model should be as close as possible to the observed examples but, at the same time, avoiding the overspecialisation and the noise as, for instance, the pattern "SABC{DFGHIM|EFGJKLN}OPT".

Activity Code	Activity Description
S	Start the task
A	Peel the carrot
B	Wash the carrot
C	Cut the carrot into pieces
D	Take a glass pot
E	Take a metal pot
F	Fill the pot with water
G	Add salt
H	Put the pot into the microwave.
I	Turn on the microwave at the maximum level (15 minutes).
J	Put the pot on the stove
K	Turn on the stove at the maximum level (10 minutes).
L	Reduce the stove at the minimum level (30 minutes).
M	Turn off the microwave.
N	Turn off the stove.
O	Take a deep bowl.
P	Add two tablespoons of soup in the dish.
T	Finish the task

Table 1: Vocabulary of activities for cooking a carrot soup.

In this paper, we propose a new inductive method whose advantages are summarised as follows:

- The method is able to identify and extract different models of a task from a reduced set of executions performed by experts. These executions are presented as sequences of activities.

- Our approach relies on a representation language based on graphs for characterising both the examples and the models. Examples and models are represented as dependency graphs, a kind of representation that simplifies an activity sequence, by only considering the consecutive dependencies directly, and representing each activity just once in the graph. Dependency graphs are hence a more concise *non-univocal* simplification of an activity sequence. Similar formalisms based on graphs can be found in the workflow learning literature (Yang et al., 2017; Günther & van der Aalst, 2007; Agrawal et al., 1998)(van der Aalst & Van Hee, 2004) with applications in bioinformatics (Yan et al., 2005), social-network analysis (Carrington et al., 2005) or web navigation (Papadimitriou et al., 2010). However, we take advantage of the properties derived from the graphs to propose an inductive method that is able to generate more than one model, each one identifying a different form of performing the task showed by the examples.

- The method is driven by two main operators: aggregation and refinement. Those operators perform the generalisation of the examples (by aggregating graphs) and the refinement of the aggregated graph (by removing some edges according to a certain threshold). Unlike other process mining methods based on graphs, our method applies the two operators repeatedly generating one model in each iteration, until all the training examples are represented by at least one of the inferred models.

- The learned models are characterised by not having disconnected nodes, so the ending activity is always reachable from the initial activity following a path in the graph; in other words, the models represent correct ways to perform the task.

- While other methods for learning tasks from a few executions get black-box models that are not comprehensible or their decisions are impossible to explain (Duan et al., 2017), our models are completely understandable by the experts since there is a one-to-one correspondence between nodes in the model and activities. Hence, the whole learning process can be audited.

We have applied our method to two real problems. The first one is a suturing task from the domain of skill training in minimally invasive surgery (MIS), which we will use as a running example. This is one of the most complicated routines to perform because of noise (Cao et al., 1996) and because surgeons can express different suture styles (Ahmidi et al., 2017). The second real application is related to cooking: the preparation of a brownie (Spriggs et al., 2009). This allows us to analyse the applicability of the model to very different domains.

The rest of this paper is organised as follows. Section 2 reviews some related work. Section 3 presents the notation we use for representing examples and models. We also define our inductive method for learning multiple models of a task. Section 4 presents the experimental results of the application of our approach to real examples taken from training in two domains: laparoscopic surgery and cooking. In Section 5 we discuss the experimental results and the strengths and limitations of our method. Finally, Section 6 outlines the conclusions and further research directions.

2. Related Work

Traditionally, the recognition of complex human activities from a set of observed data has been addressed in the area of Activity Recognition (AR) (Sukthankar et al., 2014). In AR, a distinction can be made between data-driven approaches and knowledge-driven approaches (Chen et al., 2012a).

Data-driven approaches (Hoey et al., 2010; Sánchez et al., 2008; Patterson et al., 2005; Kruger et al., 2014) are characterised by the use of supervised (e.g., Hidden Markov Models (HMM), Linear Dynamical Systems (LDSs)) and unsupervised (e.g., KNN) machine learning techniques to address the problem. These are powerful tools when facing uncertainty and temporal information but they require large datasets for learning the activities. By contrast, knowledge-driven approaches (Okeyo et al., 2011; Ye et al., 2015; Chen et al., 2012b, 2008; Bouchard et al., 2006) are characterised by their reusability, semantic clarity and a lower dependence on the training data. Activity models consist of rules that define the logic of the task and its constraints. This representation makes it possible to introduce domain knowledge. This prior knowledge can be easily translated into reusable structures, i.e., schemes, logical representation or ontologies, which are then used for reasoning about the relationships between activities, objects, temporal and spatial context.

The applicability of both types of approaches inside a real automatic supervision system has become a hot topic in recent years. Thus, data-based systems have been proposed, for instance, for assisting persons with dementia during handwashing (Hoey et al., 2010), for kitchen activities supervision (Kruger et al., 2014; Yordanova et al., 2017; Rohrbach et al., 2012; Neumann et al., 2017) and for supervision in a whole smart environment using sensor data (Twomey et al., 2016; Intille et al., 2006; Kröse et al., 2008; Crandall & Cook, 2011). Additionally, knowledge-based supervision systems try to take full advantage of other knowledge resources in contexts where accessing training data is complicated. For instance, in (Okeyo et al., 2011; Ye et al., 2015; Chen et al., 2012b) different knowledge-driven approaches are proposed to recognise the behaviour of smart home inhabitants by using ontologies. Chen et al. (2008) present a knowledge-driven framework for Smart Homes based on the Event Calculus (EC) formalism (Mueller, 2014), and Bouchard et al. (2006) present a similar idea to prevent home accidents of people that suffer from cognitive impairments (e.g., Alzheimer). The dependence on data makes data-based systems sensitive to issues such as data scarcity (it is difficult and costly to have enough training data from different users, specially when data is collected from devices such as sensors) or even the "cold start" problem: the problem that emerges when a data-driven application does not have the minimum amount of data to learn the task and operate. In addition, limitations in scalability and reusability may also arise due to difficulties in applying the learnt models from one person to another (since humans perform activities differently, they have different activity patterns). A way to deal with those problems is to apply multitask learning (Caruana, 1997) by considering each user as a task (Liu et al., 2015; Sun et al., 2012) or considering each activity as a task and solving multiple tasks by exploiting the commonalities and differences across them (Peng et al., 2018). An alternative is to perform transfer-based activity recognition (Cook et al., 2013) by transferring the knowledge learnt from source domains (users, body parts, devices, etc.) to a target domain (Van Kasteren et al., 2010; Chen et al., 2019; Ding et al., 2019). Above all, despite being possible to describe a scene by recognising the sequence of actions, in many cases, this is meaningless for understanding the task by itself, so a high-level reasoning expressing the domain knowledge is required to those effects. On the other hand, knowledge-based systems have limitations when handling noisy and uncertain data, as well as managing temporal information. In some cases, these models could be viewed as quite rigid and incomplete (Chen et al., 2012a).

The issue of determining the process behind a human task has been tackled in many ways. An interesting line of work is to create a completely connected model and then learn the transitions from a set of training runs. Probabilistic models are often used, such as Hidden Markov Models (HMMs) (Boger et al., 2005; Kalra et al., 2013; Duong et al., 2009) or Dynamic and Naïve Bayes networks (Baker et al., 2009; Dai et al., 2008; Oh et al., 2014). The learnt model is better adapted to the real behaviour of the process since it is based on evidence. However it suffers from the same drawbacks of data-based approaches since a large evidence is required to learn the transition probabilities. Additionally, in complex domains the model can end up in an explosion of possible states increasing the computational cost of these approaches considerably (Sadilek & Kautz, 2012; Rosen et al., 2002).

Another two areas that address the challenge of inferring a process model directly from observing demonstrative examples are imitation learning and process mining. The objective of imitation learning (Hussein et al., 2017) is that a machine (robots in many cases) can learn the necessary movements (i.e., policy) to solve a simple task through examples performed by a human expert while the machine observes it. The current trend in imitation learning focuses on the use of deep learning techniques (Xu et al., 2018; Duan et al., 2017; Finn et al., 2017). Although they have given surprising results, the tasks they are able to learn and solve are still very basic (e.g., stacking boxes). In addition, the fact of using deep learning hinders the traceability and analysis of the knowledge acquired by the system. On the other hand, process mining (Agrawal et al., 1998; van der Aalst, 2016) includes a family of techniques that support the analysis of business processes from some observations on how they are currently being executed. More concretely, process discovery focuses on learning process models from traces of activities (i.e., real process executions or *event log*). The generated model represents the main flow of activities in the process at hand. Process mining has been mainly applied to business management (van der Aalst et al., 2007) but also to other fields such as healthcare (Mans et al., 2008; Yang et al., 2017). Although the previous techniques are able to explain the processes in an organisation, they cannot be directly applied for the purpose of supervision: the models may contain non-essential activities for carrying out the process and they are not able to capture the different ways (if any) of performing a process since only one model is built.

In this paper we cover two application domains: skill training in minimally invasive surgery and cooking. We believe these two domains are representative of situations where there are more than one possible way of performing a task, either because of symmetries (left and right hands in surgery) or indistinct sequences (ingredient order). Other datasets in the cooking domain, such as the salad preparation dataset (Stein & McKenna, 2013) or the sandwich preparation dataset (Spriggs et al., 2009), present a huge variability in executions, not really displaying a short number of patterns, and applying pattern extraction to them would not be very meaningful (probably ending up with a pattern per example). Our method is designed for well-structured high-to-medium level activity sequences. This occurs in tasks such as those described in the case studies: surgery and recipes.

There are domains that may show one or a small number of patterns, but they are usually found in the area of activity recognition (van Kasteren et al., 2011; Voulodimos et al., 2012; Liu et al., 2015). These datasets usually contain low-level data extracted directly from devices such as sensors or smartphones. Therefore, in order to apply our method to these domains we would need to add a first phase to extract and determine the events, and then a second phase (using the proposed algorithm) to learn the sequence of events. The quality of the final process mining (and the number of patterns found) would be strongly determined by the framing of the activity recognition stage (and the choice of method there) and not only by our method, making the comparison of results prone to too many confounding factors. Nevertheless, beyond the scope of the paper, we see a lot of potential in the future to explore and apply our method in the best possible combinations with methods for activity recognition.

3. A graph-based method for inferring Multiple Models of a Task

In this section, we present a method to learn the different ways of carrying out a task from several executions of the task performed by experts. We consider a task execution as a sequence of activities that are realised consecutively with a start and an end. Following that sequence, the task is successfully completed. Formally, we define $\mathcal{A}$ as the set of activities (vocabulary) that can be used to accomplish the task, being an execution example a finite sequence of these activities $\delta = (a_1, a_2, \ldots, a_n), a_i \in \mathcal{A}$. We denote the full set of sequences provided to the system as Δ.

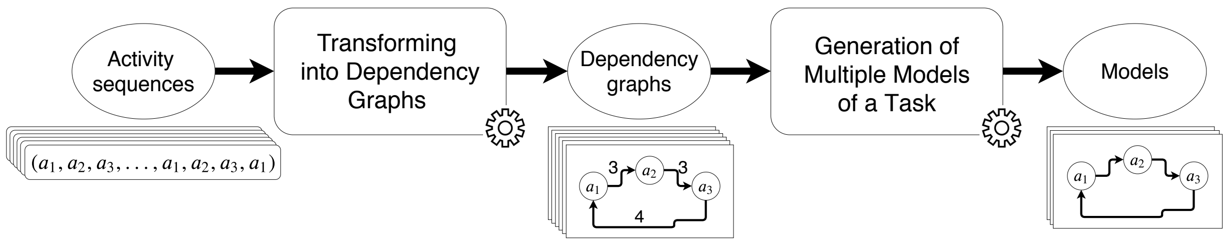
Working directly with sequences has several limitations in order to learn and express, in an intuitive and simple way, how a task should be realised. Firstly, if the sequence is too long, it becomes extremely difficult to understand the flow of activities that is taking place. Note that the main aim is to obtain a model of the task that could be used for training and/or supervising non-expert apprentices. Therefore, it is crucial that the learnt model can be easily interpreted by a human. Secondly, the problem of learning from task executions (especially when the tasks are complicated to execute because of the skills they require) is that the sequences may contain infrequent activities or noise. In our case, noise are those activities that are not really needed to complete the task[2]. It may be complicated to detect during the learning process whether the elimination of these noisy activities from the sequences leads to invalid models that are not able to complete the task, because the same activity can be essential in one part of the task and unnecessary in another part. Therefore, the use of a graph-based representation language to compactly simplify the sequences of activities as dependency graphs helps in this sense (see Section 3.2). In this case, the possible loss of information due to the change of representation is clearly compensated because dependency graphs greatly facilitate the process of identifying which activities are essential to perform a task (and therefore will be part of the models), which activities are dispensable because they are non-essential, as well as the formal confirmation that a model really express a correct way to fulfil the task.

The general pipeline of our approach is shown in Figure 1. In the first step, each sequence of activities is simplify into a dependency graph. The second step is the algorithm that infers the different models.

3.1. Graph-based formulation: notation and definitions

A dependency graph is a labelled directed graph $G = (V, E)$, where V is a set of labelled vertices, and $E \subseteq V \times V$ is a set of weighted directed edges, such that each edge has a weight given by a weight function $\omega : E \to \mathbb{N}_{\geq 1}$. The labels of the vertices belong to a finite set of labels $\mathcal{L} = \{l_1, \ldots, l_n\}$, where each l_i, $1 \leq i \leq n$, denotes an activity $a_i \in \mathcal{A}$. V contains two special vertices, v_S and v_F, that represent two synthetic activities denoting the starting and ending of the task. The two synthetic activities are implicitly placed at the beginning

[2]In other approximations, noise is considered any missing activity or any non-executed activity inserted in a sequence δ. However, the treatment of this kind of noise is out of the scope of the problem discussed in this paper.



Activity sequences
$(a_1, a_2, a_3, \dots, a_1, a_2, a_3, a_1)$
Transforming into Dependency Graphs
Dependency graphs
3
3
4
a_1
a_2
a_3
Generation of Multiple Models of a Task
Models

Figure 1: The pipeline to learn the task models.

and end of each activity sequence, respectively. Analogously, the set of labels $\mathcal{L}$ is extended to include two special labels S and F associated to the vertices v_S and v_F, respectively. For the sake of readability, given a graph G, $V(G)$ and $E(G)$ denote the vertex and edge sets of G, and ω_G denotes its weight function. Given an edge $e = (v_i, v_j)$, v_i and v_j are the source and target vertices of e, which can be retrieved using the functions σ and τ, respectively (i.e., $\sigma(e) = v_i$ and $\tau(e) = v_j$). Given a set of graphs $D = \{G_1, G_2, \dots, G_n\}$ and an edge e, we define D_e as the subset of D that contains e, that is, $D_e = \{G_i \in D \mid e \in E(G_i)\}$.

In what follows we introduce the notions of walk and validity, and three types of distinctive graphs that are used by the learning algorithm.

Definition 1. (Walk). Given a dependency graph $G = (V, E)$, and two vertices $\{v_i, v_j\} \in V$, a walk $w(v_i, v_j)$ between v_i and v_j is any sequence of directed edges $(e_1, e_2 \dots, e_n)$ from E, such that $\sigma(e_1) = v_i$, $\tau(e_n) = v_j$, and for any pair of consecutive edges e_k and e_{k+1}, $\tau(e_k) = \sigma(e_{k+1})$. A walk between v_S and v_F is called a complete walk, denoted as $\hat{w}$.

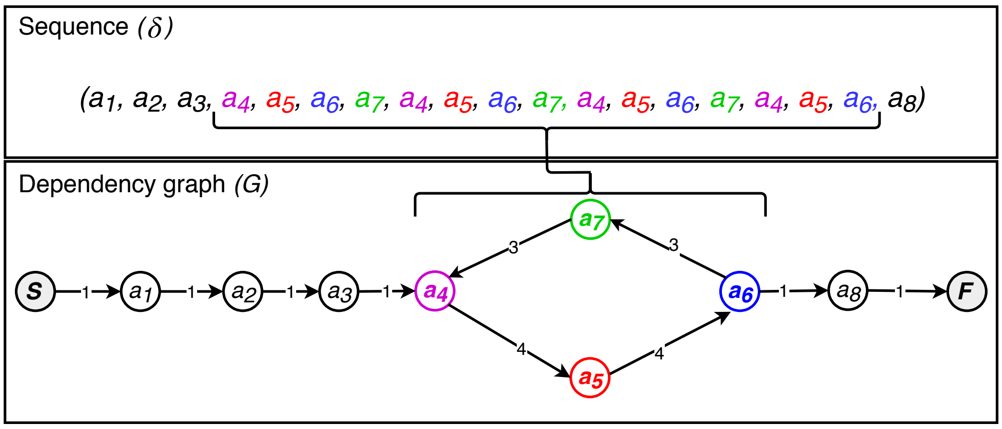
Definition 2. (Validity).Given a dependency graph $G = (V, E)$, we say that G is valid if $V_S = V_F$, where $V_S = \{v_i \in V \backslash \{v_S, v_F\} \mid \exists w(v_S, v_i)\}$ and $V_F = \{v_i \in V \backslash \{v_S, v_F\} \mid \exists w(v_i, v_F)\}$.

Definition 3. (Aggregated graph). Given a set of dependency graphs $D = \{G_1, G_2, \dots, G_n\}$, the aggregation of D is the graph $G^+ = (V^+, E^+)$, such that $V^+ = \bigcup_{G_i \in D} V(G_i)$, $E^+ = \bigcup_{G_i \in D} E(G_i)$, and $\forall e \in E^+, \omega_{G^+}(e) = \sum_{G_i \in D_e} \omega_{G_i}(e)$. We call G^+ as the *aggregated graph.*

Definition 4. (Intersected graph). Let $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ be two dependency graphs, the intersection of G_1 and G_2 is defined as $G^\cap = G_1 \cap G_2 = (V_1 \cap V_2, E_1 \cap E_2)$, where $\forall e \in E_1 \cap E_2$, $\omega_{G^\cap}(e) = min(\omega_{G_1}(e), \omega_{G_2}(e))$.

Definition 5. (Threshold graph). Given a dependency graph $G = (V, E)$ and a threshold $\theta \in \mathbb{N}_{\geq 1}$, we define the *threshold graph* $G_\theta = (V_\theta, E_\theta)$ as the subgraph of G such that $E_\theta = \{e \in E \mid \omega_G(e) \geq \theta\}$ and $V_\theta = \{\sigma(e) \mid e \in E_\theta\} \cup \{\tau(e) \mid e \in E_\theta\}$.

Definition 6. (Overlap).Given two dependency graphs G_1 and G_2, we say that G_1 and G_2 overlap if their intersection $G^\cap = G_1 \cap G_2$ is valid.



Sequence (δ)
(a1, a2, a3, a4, a5, a6, a7, a4, a5, a6, a7, a4, a5, a6, a7, a4, a5, a6, a8)
Dependency graph (G)
S
a1
a2
a3
a4
a5
a6
a7
a8
F
1
3
4

Figure 2: A sequence of activities (on the top) expressed as a dependency graph (on the bottom). The vertices corresponding to the synthetic activities (S and F) are highlighted in bold and shadowed, and the activities that appear more than once are depicted in colour.

Remark. From the above definitions we remark that: (1) there are complete walks in a valid dependency graph; (2) if an intersected graph $G^{\cap} = G_1 \cap G_2$ is valid then it contains all the common complete walks of G_1 and G_2; (3) the application of a threshold does not preserve the graph validity property.

In what follows we use the labels of the vertices to denote them.

3.2. Data formalisation: from activity sequences to dependency graphs

In this section, we describe how to convert an activity sequence $\delta = (a_1, \ldots, a_n)$ into a dependency graph G. Firstly, $V(G)$ is created containing v_S, v_F and as many vertices v_i as different activities a_i are in δ with labels $l_i = a_i$. $E(G)$ is also initialised by containing the directed edges (S, a_1) and (a_n, F) with weights equal to 1. Then, for each pair of consecutive activities $(a_i, a_j) \in \delta$, if the edge $e = (a_i, a_j)$ already belongs to $E(G)$, then its weight is increased $\omega_G(e) = \omega_G(e) + 1$; otherwise, e is added to $E(G)$ with weight $\omega_G(e) = 1$. Figure 2 illustrates this conversion process. On the top, it is shown an activity sequence of length 19 composed by 8 different activities (a_1 to a_8), some of them appearing more than once (marked in colour). This sequence is expressed as the dependency graph on the bottom of this figure, which is formed by 10 vertices and 10 edges. The labels on the edges indicate their weights. For instance, the weight of edge (a_4, a_5) is 4 because the activity a_4 appears followed by the activity a_5 four times in the sequence.

Note that a dependency graph is a representation more concise than the original activity sequence. That means that different sequences can produce the same dependency graph. This can happen when there are activities that are performed more than once along the sequence. This could imply losing some information about the order in that the activities have been done when the sequence turns into a dependency graph. For instance, the sequences $\delta_1 = (a_1, a_2, a_3, a_4, a_2, a_4, a_2, a_4, a_5)$ and $\delta_2 = (a_1, a_2, a_4, a_2, a_4, a_2, a_3, a_4, a_5)$ generate the same dependency graph. Thus, by looking at the graph we can only say that activity

a_3 is performed after activity a_2 but not when (after doing a_2 for the first, the second or the third time). This fact is known as “representational bias” in the area of process mining (van der Aalst (2016)) or “language bias” in the inductive logic programming field (Adé et al. (1995)), and it refers to the implicit choices that restrict the syntax of examples and hypothesis. By contrast, one of the main benefits of the language bias is that it reduces the search space of possible models. In our case, the graph representation allows us to search the models by systematically applying the graph operations defined in Section 3.1.

3.3. Mining Multiple Models from dependency graphs

In this section we describe a new method for inducing multiple models from a set of dependency graphs $\mathcal{G} = \{G_1, G_2, \ldots, G_n\}$. A model M is a dependency graph that satisfies the following conditions:

- M is valid and, consequently, there exists in M (at least) one walk from S to F. In terms of the original task, this means that such a walk indicates the activities that are needed to perform the task.

- M has to overlap at least one dependency graph in $\mathcal{G}$. Newly, in terms of the original task, this means that there exits at least one expert execution that has performed the same activities included in M. In other words, M captures one of the ways of solving the task according to the experts.

Additionally, the presence of noise in M (non-essential activities for the task) should be minimised as much as possible, in order to obtain the desired trade-off between generality and specificity.

The MMDG algorithm (Algorithm 1) combines a generalisation operator (graph aggregation) with a refinement operator. After applying both operators, the dependency graphs that overlap (Definition 6) with the induced model are removed. This process is repeated using the remaining graphs until all the dependency graphs overlap with one of the models. In this way, we are able to infer the different styles of solving the task showed by the experts in their executions. The MMDG algorithm is inspired by sequential covering strategies of rule learning (Fürnkranz, 1999).

More concretely, the MMDG algorithm works as follows. First, the most general graph is generated by aggregation (Line 4). This aggregated graph contain all the vertices and edges included in the dependency graphs $G_i \in \mathcal{G}$. Clearly, this aggregated graph is too much general: (1) it contains not only valid walks but other new walks (not included in any G_i) by combining partial walks from several G_i (that probably do not represent a correct way to execute the task), and (2) it is the noisiest graph of all that can be generated from $\mathcal{G}$. Hence, G^+ must be refined below (Line 6). This refinement returns one model M and the set of dependency graphs $\bar{\mathcal{G}}$ that overlap with M. Then, the set of dependency graphs $\bar{\mathcal{G}}$ that overlap with the refined graph M are removed from $\mathcal{G}$ (Line 7) and M is added as a part of the solution (Line 8). Then, the next iteration of the algorithm starts with the remaining examples, creating a new aggregated graph and so on. This process is repeated until complete the solution.

Algorithm 1 MMDG ALGORITHM

```
Require: a set of dependency graph: $\mathcal{G} = \{G_1, G_2, \dots, G_n\}$.
Ensure: a set of models: $\mathcal{M}$.
 1: $\mathcal{M} \leftarrow \emptyset$.
 2: // Iterate over the set of graphs until each $G_i$ has a $M$ that overlaps with it.
 3: while $\mathcal{G} \neq \emptyset$ do
 4:     $G^+ \leftarrow$ AggregateGraphs($\mathcal{G}$).
 5:     // $\bar{\mathcal{G}}$ is the set of overlapped dependency graphs.
 6:     $\{M, \bar{\mathcal{G}}\} \leftarrow$ REFINEMENT($G^+$, $\infty$, $\mathcal{G}$).
 7:     $\mathcal{G} \leftarrow \mathcal{G} \setminus \bar{\mathcal{G}}$.
 8:     $\mathcal{M} \leftarrow \mathcal{M} \cup \{M\}$.
 9: end while
10: return $\mathcal{M}$
```

The refinement of the aggregated graph G^+ (Algorithm 2) is performed by repeatedly applying a threshold θ to G^+, that is, by removing edges according to their weights. Hence, we vary the threshold decreasingly starting from a maximum value, so the first graphs G^+_θ that are generated are usually very simple as many edges are removed. The application of a threshold has the aim to remove the non-essential activities from G^+. Progressively, we reduce the threshold and apply it until get a G^+_θ that satisfies the conditions to be a model: it is valid and overlaps at least with a dependency graph G_i. Figure 3 illustrates the application of a threshold to an aggregated graph. On the top, we see an aggregated graph G^+ formed by $N = 10$ dependency graph. When we use a high threshold (i.e., $\theta = 9$), the resulting threshold graph (shown in the middle of the figure) is not valid. Meanwhile, by taking a lower threshold (for instance, $\theta = 7$), we obtain a valid threshold graph (shown on the bottom) that represents a way to completely perform the task. It is worth noting that some activities that are rarely performed in these examples (denoted by the low weight of their incoming and outgoing edges in G^+), such as a_2 or a_6, are not included in $G^+_{\theta=7}$.

The input of the REFINEMENT algorithm (Algorithm 2) is G^+, an initial threshold θ, and $\mathcal{G}$ (the set of dependency graphs used to generate G^+). In the case the initial threshold is ∞, the algorithm sets the θ value at the minimum weight that guaranties that G^+_θ could be valid (Lines 3-9). The θ value is firstly used on G^+ to discard those edges whose weight is below that value (Line 10). There are two reasons why this θ may be inappropriate to generate a model: (1) it produces an invalid G^+_θ, or (2) G^+_θ is valid but does not overlap with any G_i. In the first case, the θ value is repeatedly decreased[3] until a valid G^+_θ is found[4] (Lines 12-15). For case (2), the next lower θ value is selected as in the first case, and a

[3]For efficiency reasons, the next value for θ is selected from the weights of the edges in E^+, assuming E^+ ordered in decreasing order of their edges weights.

[4]Observe that selecting the θ values in decreasing order of the weights in E^+ assures the termination of the algorithm 2, since in the worst case the minimum weight in E^+ is taken as θ, which means that no edges are removed from the aggregated graph, and therefore G^+_θ is a valid graph that overlaps with all the dependency graphs used to construct it.

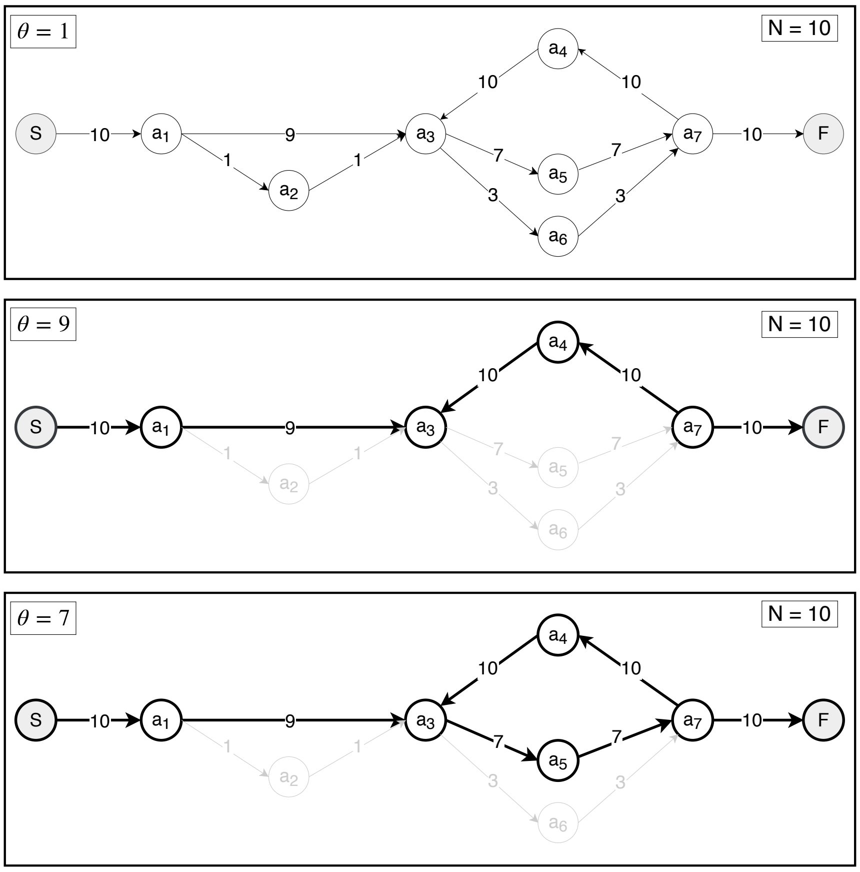


Figure 3: Effect of applying different thresholds for refining a G^+ formed by the aggregation of 10 dependency graphs (shown on the top). Note that the refinement procedure keeps all the edges of G^+ whose weight is greater than θ.

recursive call is performed (Lines 24-27). According to Definition 6, to check whether a threshold graph G^+_θ overlaps with a dependency graph G_i, we verify the validity of their intersection $G_i \cap G^+_\theta$ (Lines 17-22).

Finally, when a valid G^+_θ overlaps with one or more examples, the algorithm returns G^+_θ as a model, and the list of G_i that overlap with it (Line 28).

Figure 4 shows an example of how the overlap between graphs is checked. From top to bottom, we see one threshold graph G^+_θ, two dependency graphs G_1 and G_2, and the intersected graphs $G_1 \cap G^+_\theta$ (left) and $G_2 \cap G^+_\theta$ (right). Note that $G_1 \cap G^+_\theta$ is not a valid dependency graph because it does not contain any complete walk, which means that G^+_θ does not capture the way of performing the task showed by the complete walk in G_1. However, the sets V_S and V_F in $G_2 \cap G^+_\theta$ are equal, $V_S = V_F = \{a_1, a_3, a_5, a_8, a_7, a_4\}$, and hence G^+_θ overlaps with G_2.

Algorithm 2 Refinement Algorithm

Require: an aggregated graph: $G^+ = (V^+, E^+)$; a threshold value: θ; a set of dependency graph: $\mathcal{G} = \{G_1, G_2, \ldots, G_n\}$.

Ensure: a model M: G^+_θ; a set of overlapped examples: $\bar{\mathcal{G}}$.

```
1:  $\bar{\mathcal{G}} \leftarrow \emptyset$.
2:  // Find the minimum weight to begin and end a $\hat{w}$.
3:  if $\theta$ is $\infty$ then
4:      $v_S \leftarrow$ GetStartVertex($V^+$).
5:      $v_F \leftarrow$ GetEndVertex($V^+$).
6:      $starting\_weight \leftarrow$ MaxEdgeWeight($E^+$, $v_S$).
7:      $ending\_weight \leftarrow$ MaxEdgeWeight($E^+$, $v_F$).
8:      $\theta \leftarrow \min(starting\_weight, ending\_weight)$.
9:  end if
10: $G^+_\theta \leftarrow$ ApplyThreshold($G^+$, $\theta$).
11: // Adjust the $\theta$ value in order to get a valid $G^+_\theta$.
12: while not IsValid($G^+_\theta$) do
13:     $\theta \leftarrow$ NextWeightBelowThreshold($E^+$, $\theta$).
14:     $G^+_\theta \leftarrow$ ApplyThreshold($G^+$, $\theta$).
15: end while
16: // Check whether the valid $G^+_\theta$ also overlaps at least one $G \in \mathcal{G}$.
17: for each $G$ in $\mathcal{G}$ do
18:     $G^\cap \leftarrow G \cap G^+_\theta$.
19:     if IsValid($G^\cap$) then
20:         $\bar{\mathcal{G}} \leftarrow \bar{\mathcal{G}} \cup G$.
21:     end if
22: end for
23: // Do a recursive call with a lower $\theta$ if any $G$ overlaps with the current $G^+_\theta$.
24: if $\bar{\mathcal{G}}$ is $\emptyset$ then
25:     $\theta \leftarrow$ NextWeightBelowThreshold($E^+$, $\theta$).
26:     $\{G^+_\theta, \bar{\mathcal{G}}\} \leftarrow$ Refinement($G^+$, $\theta$, $\mathcal{G}$).
27: end if
28: return $\{G^+_\theta, \bar{\mathcal{G}}\}$
```

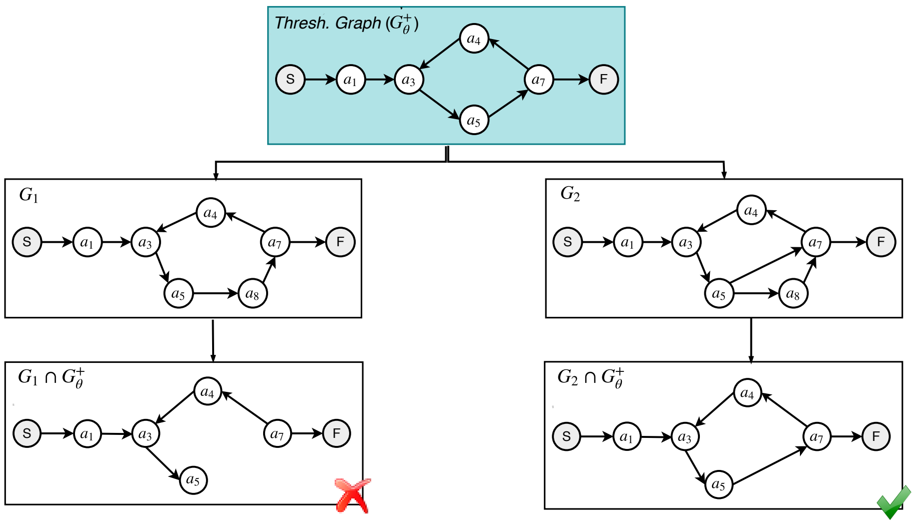


Figure 4: Overlap checking between a threshold graph G_θ^+ and two dependency graphs G_1 and G_2. There is no overlapping between G_θ^+ and G_1 because their intersection $G_1 \cap G_\theta^+$ (on the left bottom) does not hold the validity constraint ($V_S \neq V_F$). In contrast, G_θ^+ overlaps with G_2 because their intersected graph $G_2 \cap G_\theta^+$ (on the right bottom) is valid ($V_S = V_F$).

4. Experimental evaluation

We will analyse the quality of our method through a set of metrics that allow us to determine how well the models capture the styles of performing a task observed in the expert executions (expressed as dependency graphs). More concretely, we will consider the following two quality measures: fitness and simplicity. Given a set of dependency graphs $\mathcal{G}$ and a set of the models $\{M_i\}$, the fitness of each model M_i is the number of dependency graphs of $\mathcal{G}$ with which M_i overlaps. Note that if some dependency graphs overlap with a certain model is because they share a similar way of solving the task. Hence, the fitness of a model is an indicator of how well the model captures such a way to execute the task. To compute the fitness of a solution (the set of inferred models), the models are applied in the order in that are generated. Lastly, the simplicity (or, alternatively, complexity) of a model is measured in terms of the number of edges and vertices that compose it. Fitness and simplicity are well-known quality metrics of process discovery algorithms (Buijs et al., 2012). They are related to other state-of-the-art evaluation metrics in data science (Geng & Hamilton, 2006): *Generality/Coverage* and *Conciseness* measures, respectively. Additionally, the generality/coverage relation is employed in concept learning, rule-learning and inductive logic programming to determine the examples that are matched by a hypothesis (model) (Raedt, 2010).

To experimentally evaluate our approach, we implemented a prototype using the programming language R[5]. The full code and data used for the experiments can be found in

[5]`https://www.r-project.org/`

a github repository[6]. We chose a challenging test-bed domain as running example: a typical suturing procedure in advanced surgeries like Laparoscopic Surgery. In laparoscopic exercises, one can find certain activities that can be considered as support manoeuvres. Specifically, the surgical tasks that require many regrasping movements entail additional motions such as positioning or the tool reorientation which are not always really needed to perform the task. Additionally, the surgeon's idiosyncrasy and the diversity of surgery "styles" are another factors to be taken into account if we are interested in learning models in this domain. Our inductive method is able to account for all these factors during the learning of the models. Finally, in order to test the versatility of the method, we have also applied it in a different domain, cooking brownies.

4.1. Case Study 1: Surgical data

There are not many public medical datasets specialised in laparoscopic exercises. Among the few available, possibly the most popular and complete is the JHU-ISI Gesture and Skill Assessment Working Set (JIGSAWS) dataset (Gao et al., 2014), which has been registered by using the surgery system robotics *da Vinci*. Originally conceived for developing applications focused on the surgical motion analysis and the automatic skill assessment, this dataset has become a public benchmark for evaluating the performance of the state-of-the-art methods in surgical activity recognition (Ahmidi et al., 2017). Comparing with other human tasks, including surgery domain, the activities involved in suturing entail more complexity and diversity of movements than other training routines (Cao et al., 1996).

We used the suturing executions provided by JIGSAWS dataset to learn the models from the different ways in which this task can be conducted. A simple suturing routine is divided into three phases. Firstly, the surgeon has to reach the needle and move it at the corresponding dot on the tissue. Then, the main phase of the exercise is performed: the suturing cycles. Figure 5 shows this phase. For each cycle iteration (loop), the surgeon must push the needle against the next dot and take it out on the other side of the incision (Frame 1). The extraction of the needle must be carried out with the other hand (Frame 2). Then, the needle is transferred to the first hand again (Frame 3). Once the suturing cycles are completed, the last phase consists in laying the needle down at the finishing mark on the tissue. Concretely, for the experiments we chose a 4-throw suturing procedure, where the digit denotes the number of loops required to perform the suture.

In this context, an activity represents an atomic surgical gesture with a meaningful outcome and it is annotated following a specific activity language of this domain (Table 2). Hence, each activity transcription includes the name of the gesture, and the start and end frames in the video. Each performance contains from 15 to 20 activities per recording. It is necessary to mention that the activities of each task performance are annotated in chronological order of execution and there are not overlapping in time. Therefore, we consider each execution as a time-ordered sequence of labelled activities that a surgeon performed to completely fulfil the suturing task. Thus, we have preprocessed the transcription files to remove the time information and, then we have converted the activity sequences into

[6]https://github.com/DNC87/MiningMultipleDependencyGraphs

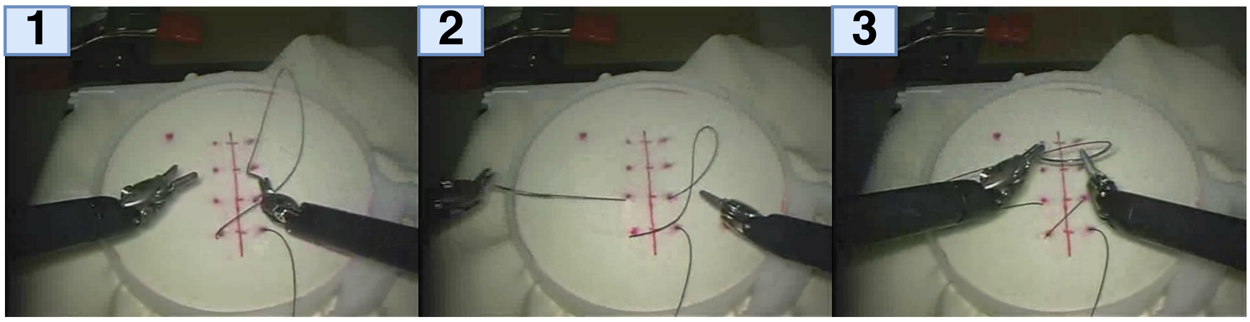
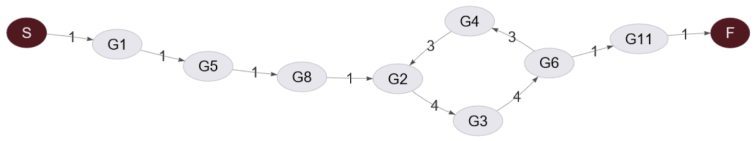


Figure 5: Example of a suturing cycle represented by frame steps. From left to right, frame *(1)* shows the needle insertion in the input dot on the tissue, frame *(2)* represents the needle grasping on the output dot from the other side of the tissue, and frame *(3)* shows the needle transferring from left tool to the right tool. Captures obtained from JIGSAWS dataset (Gao et al., 2014).

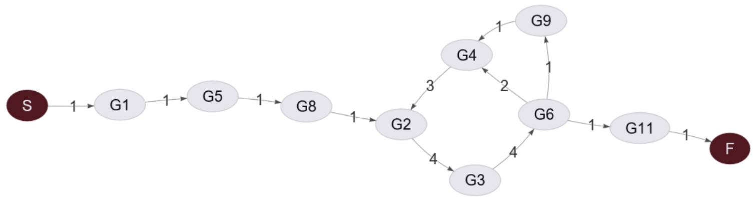
ID	Description
G1	Reaching for needle with right hand
G2	Positioning needle
G3	Pushing needle through tissue
G4	Transferring needle from left to right
G5	Moving to center with needle in grip
G6	Pulling suture with left hand
G7	Pulling suture with right hand
G8	Orienting needle
G9	Using right hand to help tighten suture
G10	Loosening more suture
G11	Dropping suture at end and moving to end points
G12	Reaching for needle with left hand
G13	Making C loop around right hand
G14	Reaching for suture with right hand
G15	Pulling suture with both hands

Table 2: Gesture vocabulary from JIGSAWS dataset (Gao et al., 2014). We consider those surgical gestures as activities throughout our case study.

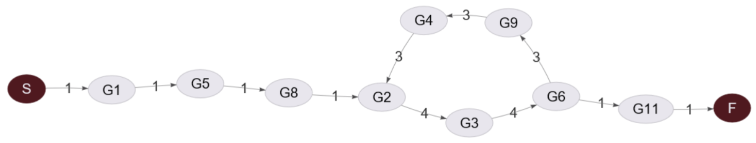
dependency graphs. Figure 6 shows three trials extracted from the JIGSAWS dataset expressed as dependency graphs. As can be seen, the three graphs describe the suturing task composed by the needle preparation phase, the suturing loop, and the ending move where the needle is released. Notwithstanding the similarities, we can identify several differences between them. Thus, trials *(a)* and *(b)* seem very similar; however, there is an additional gesture ($G9$) in *(b)* that is not in *(a)*. This is the kind of support gesture we were referring to previously. Specifically, the surgeon has used the right tool to help tighten suture. Despite these manoeuvres are perfectly valid in the surgical domain, they may not be really necessary (as in trial *(b)*), so they may be considered as noise. A different fact is when these supporting gestures are embedded inside the cycle as a routine activity as trial *(c)* shows,



(a) Suturing example 1



(b) Suturing example 2



(c) Suturing example 3

Figure 6: Three trials from different surgeons expressed as dependency graphs. The differences reside in how each surgeon has performed the suturing loop. The labels of the vertices are provided in Table 2.

which constitutes a way of performing the suturing task different of that exhibited by trials *(a)* and *(b)*.

Finally, Table 3 shows the information about the executions or "trials" (term used in the dataset to identify a task execution) provided by the dataset. To improve the readability of this table, the columns represent trials and the rows are their description. A trial is identified by a code, formed by the surgeon identifier (a letter from B to I) and the order of her trial (e.g. H003 means that surgeon H has recorded her trial number 003). As can be seen, eight surgeons performed this task 5 times which makes a total of 40 trials. The rows in the table show the metadata that JIGSAWS provided for describing each trial: the global rating score (GRS[7]) which measures the technical skill over the entire trial; the surgeon's level of expertise in robotic surgery measured in hours of practice (i.e., E-expert (>100hrs), I-intermediate (10-100hrs), N-novice (<10hrs)), and the scoring quartile that we added by stratifying the scores into quartiles. All of this information will be used to analyse the performance of our method for different trials selection criteria.

4.1.1. Experimental Setup

The input dependency graphs have been divided in two groups: the training and the test sets. Given that we have few examples, we prepared three exploratory scenarios applying different criteria to select the training graphs and analysed the impact of the training data

[7]More information about this measure in (Gao et al., 2014)

Trial	B001	B002	B003	B004	B005	C001	C002	C003	C004	C005	D001	D002	D003	D004	D005	E001	E002	E003	E004	E005
Global Rating Score (GRS)	13	17	14	10	12	26	20	26	30	17	14	18	14	8	15	17	20	19	19	19
Surgeon expertise (in robotic surgery)	N	N	N	N	N	I	I	I	I	I	E	E	E	E	E	E	E	E	E	E
Scoring Quartile	Q4	Q3	Q4	Q4	Q4	Q1	Q2	Q1	Q1	Q3	Q4	Q3	Q4	Q4	Q3	Q3	Q2	Q2	Q2	Q2

Trial	F001	F002	F003	F004	F005	G001	G002	G003	G004	G005	H001	H003	H004	H005	I001	I002	I003	I004	I005
Global Rating Score (GRS)	24	26	29	24	29	13	18	13	21	23	14	25	19	21	17	23	17	23	19
Surgeon expertise (in robotic surgery)	I	I	I	I	I	N	N	N	N	N	N	N	N	N	N	N	N	N	N
Scoring Quartile	Q1	Q1	Q1	Q1	Q1	Q4	Q3	Q4	Q2	Q1	Q4	Q1	Q2	Q2	Q3	Q2	Q3	Q2	Q3

Table 3: Trial information for suturing task in JIGSAWS dataset Gao et al. (2014). The rows represent: the surgeon's experience in robotic surgery (measured in hours): E-expert (>100hrs), I-intermediate (10–100hrs), N-novice (<10hrs); the GRS score reported by the expert reviewers, and the GRS score in quartile partitions. The columns correspond to the trials, where the letter identifies the surgeon who performed the trial and the number denotes the trial order.

selection in the learning process. For the first scenario (Experiment 1), we split the data according the surgeon's expertise level using the trials where surgeon's tag is "E" (Expert) as training graphs. For the second and third scenarios (Experiments 2 and 3, respectively), we used the GRS score as a criterion for splitting the data. Specifically, as training data we used the dependency graphs belonging to the $Q1$ quartile for Experiment 2, and the dependency graphs belonging to the $Q1$ and $Q2$ quartiles for Experiment 3. Experiments 2 and 3 were carried out to analyse the impact of the size of the training data on the inferred models. Thereby, the number of training dependency graphs used for each experiment was $n = 10$ (Experiment 1), $n = 9$ (Experiment 2) and $n = 16$ (Experiment 3). In each of these scenarios, those dependency graphs not used for training were used as test set.

In what follows, we denote the models using Roman alphabet upper-case letters in ascending order (e.g., A_i, B_i, C_i, …), according to the order in that they are generated, and where the subindex i identifies the experiment ($i \in \{1, 2, 3\}$).

4.1.2. Experimental results

Before reporting the performance obtained by our method in the experimental evaluation, we show in Figure 7 the models learnt in Experiment 1.

As can be seen, we can clearly identify the phases of a suturing exercise in models A_1 and B_1: the needle reaching, the suture cycle and the needle releasing at the finishing mark. Although both models are very similar, we detect two important differences: B_1 incorporates the activity G_8 (i.e., reorienting the needle) before the suturing cycle and the activity G_9 (i.e., using the right hand to tighten the suture) just before releasing the needle. In contrast, in the third iteration, no edges were removed from G^+ to generate model C_1. As a consequence C_1 is a large and complex model that captures more specific ways of performing the suturing exercise.

Table 4 shows the results of the learning process and the fitness analysis with respect to the training data for the three experiments. Firstly, we observe that, by chance, 3 models were inferred in each experiment. Regarding the effect of the size of the training data set, although Experiment 3 used a number of training graphs greater than the other two

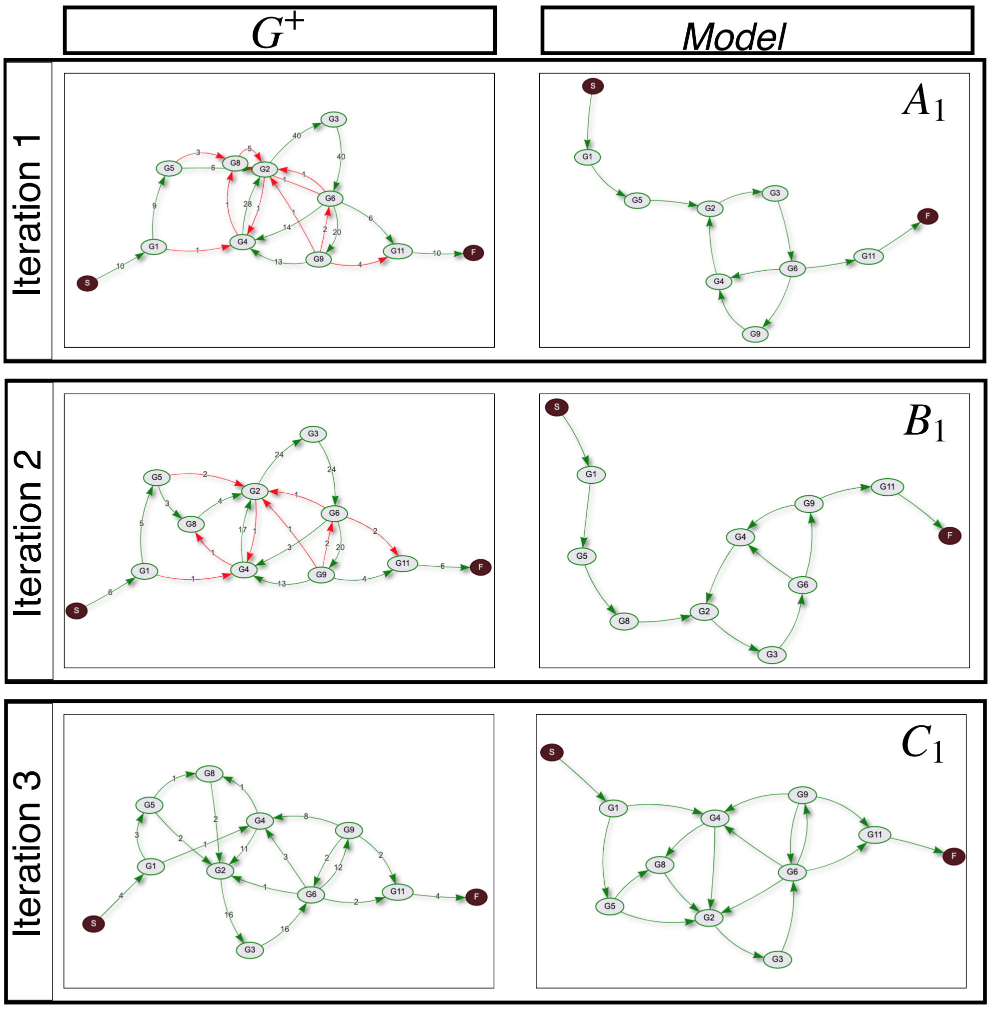


Figure 7: Models inferred from the Expert's dependency graphs (Experiment 1). Rows contain the corresponding G^+ and model generated at each iteration of the MMDG algorithm. We have coloured in green those edges in G^+ that remain in the model. By contrast, the edges coloured in red are those that are removed by the algorithm when refining G^+.

experiments, no more models were obtained. Comparing Experiments 2 and 3 (which were conceived using a selection criteria of the training data based on the score quartiles) we can conclude that the size of the training data set per se does not seem to have a direct influence on the size of the solution (number of models), but the similarity among the training data. It can be assumed that this similarity depends to a large extent on the suturing skills of the surgeon which is measured by the GRS value. Thus, the dependency graphs belonging to quartiles $Q1$ and $Q2$ should be similar in the sense that they not show any new popular way of performing the task apart from those expressed by the examples used in Experiment 2.

If we compare Experiment 1 and Experiment 2, we observe similar results between them, regarding the fitness of the models. Perhaps the most remarkable fact is that the fitness of

Experiment 1
(Experts)

ID	Model
E001	A_1
E002	A_1
E003	A_1
E005	A_1
D003	B_1
D005	B_1
D001	C_1
D002	C_1
D004	C_1
E004	C_1
Summary (n=10)	A: 4 B: 2 C: 4

Experiment 2
(GRS (Q1))

ID	Model
C001	A_2
C003	A_2
C004	A_2
F005	A_2
H003	A_2
F003	B_2
F004	B_2
F001	C_2
F002	C_2
Summary (n=9)	A: 5 B: 2 C: 2

Experiment 3
(GRS(Q1+Q2))

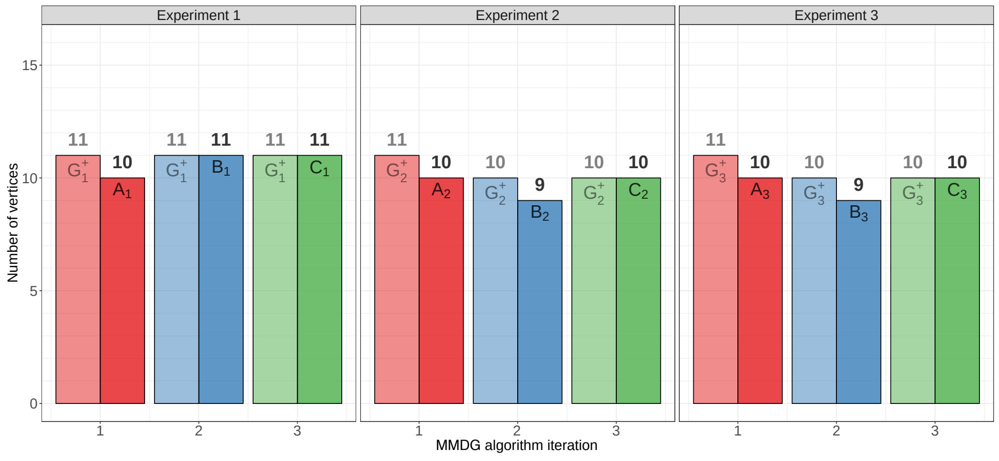
ID	Model
C001	A_3
C003	A_3
C004	A_3
F005	A_3
H003	A_3
I002	A_3
I004	A_3
C002	B_3
F003	B_3
F004	B_3
G004	B_3
G005	B_3
H005	B_3
E002	C_3
F001	C_3
F002	C_3
Summary (n=16)	A: 7 B: 6 C: 3

Table 4: For each experiment, the table contains the training dependency graphs (column ID) and the model which overlaps with them (column category). In Experiment 1, the training data corresponds to the trials performed by the experts (a total of $n = 10$ instances). In Experiments 2 and 3 the training data were selected according to their GRS score quartiles: the first quartile ($Q1$) in Experiment 2 (a total of $n = 9$ instances), and the first and the second quartiles ($Q1 + Q2$) in Experiment 3 ($n = 16$ instances in total). At the bottom of each table it is shown the fitness summary per experiment.

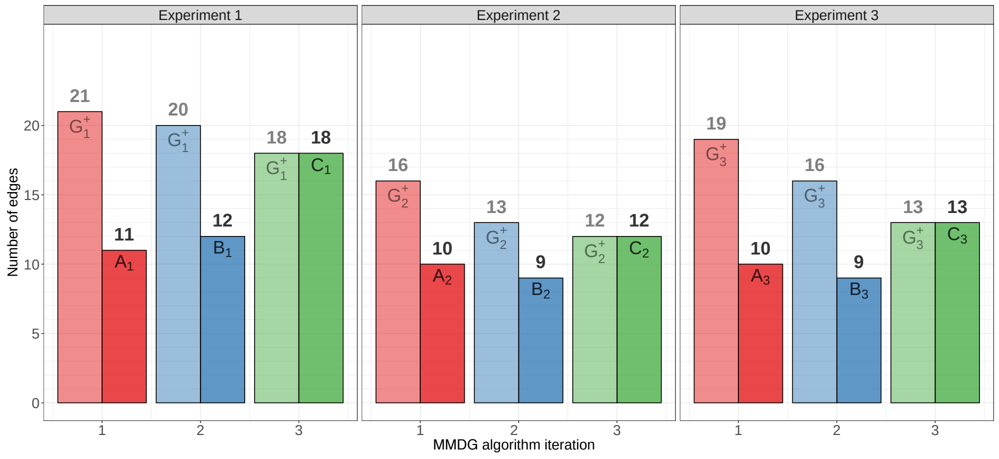
model C_1, the last model generated from the experts executions (Experiment 1) is higher than the fitness of its counterpart model C_2 from Experiment 2. This means that the training data is more uniform in Experiment 2 than in Experiment 1 (that is, there are more training graphs in Experiment 2 that follow the same styles of solving the suturing task which are captured by models A_2 and B_2).

The analysis of the simplicity of the models is depicted in Figure 8. Specifically, we show the number of vertices (top) and edges (bottom) of each induced model. The smaller the number of vertices and edges, the simpler the model is. In all the experiments, we observe that, as expected, some vertices (i.e., activities) and edges (i.e., transitions between activities) were removed during the process of refining the aggregated graphs G_i^+ to produce the models ($A_i/B_i/C_i$).

Concretely, regarding the results with respect to the vertices, it is observed a slight reduction in their number. Thus, in all the models generated in iterations 1 and 2, one vertex was filtered except in B_1. This means that one unnecessary activity to perform the task was discarded, with the subsequent positive effect on model compactness. On the other hand, the results regarding the edges show a noticeable reduction of them in models A_i and B_i (comparing them with their respective aggregated graphs). Specifically, the number of edges of A_1 decreases by 47.62% with respect to its aggregated graph. Similar decreases are observed in other models, such as A_3 (47.37%), B_3 (43.75%), A_2 (37.50%), B_1 (40.00%) and



(a) Number of vertices in G_i^+ and models A_i, B_i and C_i for each experiment.



(b) Number of edges in G_i^+ and models A_i, B_i and C_i for each experiment.

Figure 8: Number of vertices (top) and edges (bottom) of the aggregate graphs and the models generated during their refinement for Experiments 1, 2 and 3. The algorithm iterations have been distinguished using colours (iteration 1 - red, iteration 2 - blue and iteration 3 - green) and the results have been grouped by experiment.

B_2 (30.77%). All this has a positive effect on the compactness of the inferred models.

Additionally, if we compare the three experiments, we observe that using the trials executed by the experts (Experiment 1), the model G_1^+ is the aggregated graph with more edges which means that this is the graph that contains more non-essential transitions between activities. These dependency graphs are depicted in Figure 7. Analogously, the models learned in Experiment 1 were those that had the most activities and transitions, if we compare them with the respective models obtained in the other two experiments. This is further supported by the difference in the number of edges between model C_1 and models C_2 and C_3. The last models generated by the MMDG algorithm usually gather the non-essential transitions filtered in previous iterations. In this case, we observe that more edges were gathered in the case of the experts. Similar conclusions are derived from the analysis of the number of vertices in Experiment 1. Based on these results, we can conclude that the trials recorded by the experts are more noisy (they contain many actions that are non-essential for performing the task). However, our approach is able to infer models that filter that noise in all the experiments.

We also analysed how the size of the training data set affects the simplicity of the models. As mentioned above, to this end we compared Experiments 2 and 3. Thus, in the two first iterations of the MMDG algorithm the aggregated graphs had more edges in Experiment 3 than in Experiment 2. Nevertheless, the generated models had the same number of edges and vertices in both experiments. That means that the $Q2$ training graphs only contribute with transitions between activities that indeed are non-essential for the task since they were not included in the models.

Finally, to study the generalisation capacity of our models we calculated their fitness over a set of unseen dependency graphs (the test set). Table 5 shows the fitness results for the test data.

Experiment	n	A_i	%	B_i	%	C_i	%	$-_i$	%
1	29	2	6,90%	0	0,00%	16	55,17%	11	37,93%
2	30	11	36,67%	6	20,00%	1	3,33%	12	40,00%
3	23	9	39,13%	2	8,70%	5	21,74%	7	30,43%

Table 5: Fitness of the inferred models over the test data. The columns 1 and 2 show the experiment identifiers and the size of the test set, respectively. The next three column blocks correspond to the absolute (denoted as A_i, B_i and C_i) and relative (denoted as %) fitness value per model. The last block of columns ($-_i$ and %) shows the absolute and relative number of test graphs that do not overlap with any model.

In general terms, we observe that the global fitness rate (adding the fitness of all the models induced in an experiment) is in the range between 60% and 70% for all the experiments. If we observe the fitness achieved by the models induced from the experts (Experiment 1), the fitness rate of A_1 and B_1 is very low. Despite that around 62% of test graphs overlap with the models in this experiment, only 6.90% of the test data overlap with a different model to C_1. The high fitness rate of C_1 may be motivated by the fact that the last generated model usually gather more edges and activities, so it can overlap with many more examples. However, C_1 is the poorest model in terms of simplicity. By contrast, Experiment 2 shows a greater fitness rate in models A_2 and B_2, but not C_2, compared to Experiment 1. Although Experiment 2 used a smaller training set than Experiment 1, we do not observe any effect on the rate of test data that do not overlap with any model. Specifically, the rise is only about 2% over the rate reached in the Experiment 1 ($-_1 = 37.93\%$ and $-_2 = 40\%$).

On the other hand, the shift of the fitness rate towards models A and B in Experiment 2 is an interesting outcome. Models A_2 and B_2 have been more suitable to generalise the task, since they overlap with a noticeable portion of the examples not seen during the model training ($A_2 = 36.67\%$ and $B_2 = 20\%$). The fitness rates reached by models A_2 and B_2 are significantly larger than those obtained by models A_1 and B_1 (Experiment 1). However, the fitness results of Experiment 3 show a different picture. Firstly, we can see how the number of

ID	Description
A_4	crack-egg
A_12	pour-big-bowl-into-baking-pan
A_13	pour-brownie-bag-into-big-bowl
A_14	pour-oil-into-big-bowl
A_16	pour-water-into-big-bowl
A_26	spray-pam
A_27	stir-big-bowl
A_28	stir-egg
A_29	switch-on-the-oven
A_31	take-big-bowl
A_33	take-egg

Table 6: Vocabulary of the brownie cooking dataset.

test graphs that do not overlap with any model decreases from 40 % (obtained in Experiment 2) to 30.43 %, whereas most of the test graphs overlap with models A_3 and C_3. Although the fitness rate of A_3 is higher than that of A_2, the difference is only of 2.46%. Conversely, the increment in the fitness of C_3 with respect to C_2 is of 18.41%. All this corroborates the fact observed with the training data that by adding the Q_2 dependency graphs to the training set, only contributes with graphs with more non-needed activities/transitions (that is, noise), which in turn help model C_3 to fitness more graphs. Nonetheless, even in the best case of fitness (obtained by Experiment 3), almost 30% of the test graphs show way of performing the suturing task that are not gathered by the models.

4.2. Case Study 2: brownie cooking

We have applied the MMDG algorithm to another challenging problem in a different domain: cooking. This domain is especially challenging for different reasons: the examples have high variability among them and they do not contain extra information about the quality of the execution. The brownie cooking dataset (Spriggs et al., 2009) consists of 16 executions from different users and, at most, 11 different activities per execution. The scarcity of data comes not only from the number but also the length of the examples. Table 6 shows the vocabulary of the brownie cooking dataset.

Our algorithm obtains four models. Figure 9 shows the aggregated graph and the first three models. The fourth model (not shown) includes all the variability not contemplated by the three previous models, as has been described previously. With only these 16 examples, MMDG can extract three ways of cooking the brownie that allows for the understanding and supervising new executions of the task. As can be seen in Figure 9, as the threshold decreases, the variability of the model increases, giving rise to increasingly complex graphs. The model obtained in the first iteration and corresponding to a threshold of 7 (Figure 9.b) practically represents a sequence of activities and would correspond to a strict monitoring of the recipe. In the second model obtained with a threshold of 4 (Figure 9.c), once the egg has been introduced into the bowl (which coincides with the first steps of the simplest model), the model has presented some variations with respect to strict monitoring of the recipe. Finally, the third model obtained with a threshold of 2 (Figure 9.d) is the most complex and presents execution variability from the very first moment.

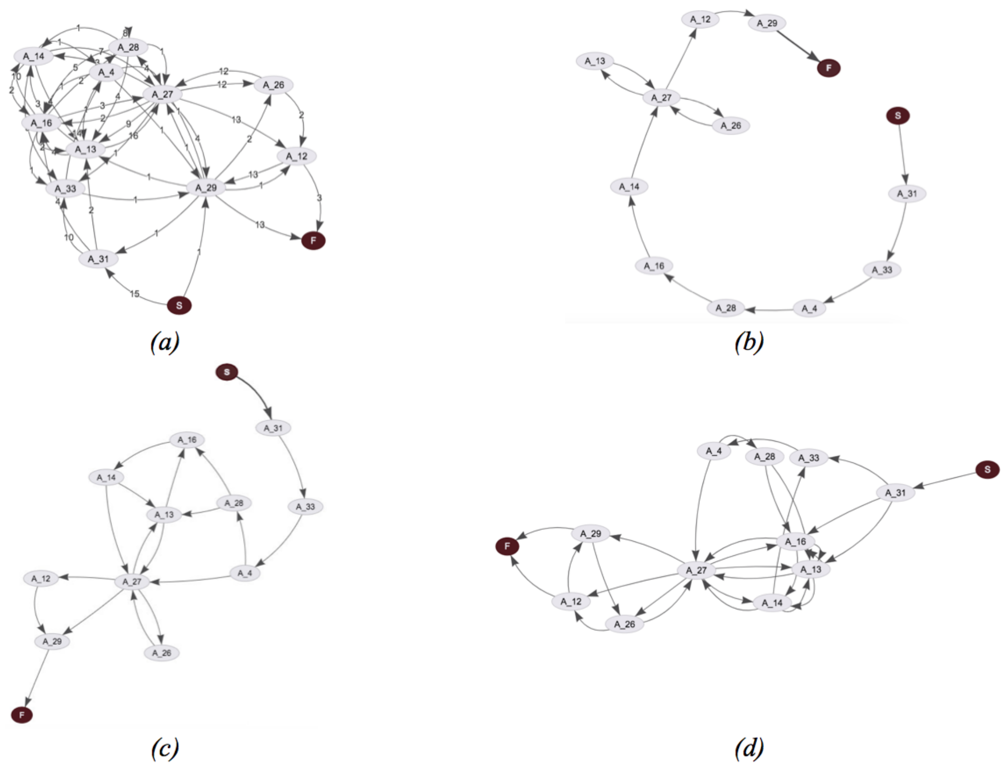


Figure 9: Brownie example. (a) aggregated graph with all examples, (b) model generated on iteration 1 and threshold 7, (c) model generated on iteration 2 and threshold 4, and (d) model generated on iteration 3 and threshold 2.

The analysis of the simplicity of the models of the brownie example is depicted in Figure 10. In this example, the induced models show a quite pronounced reduction of the number of edges in the first (marked as A) and second (marked as B) induced models. These show a reduction of 70% (14 out of 47) and 67% (20 out of 47) respectively. In the case of the third model (marked as C) the reduction is 45% (30 out of 46). The fourth model, as expected, has no edge reduction. Regarding the results with respect to the vertices, there is no reduction in any of the obtained models. This is because in these examples all the activities are present in the 16 examples and only the transition between executions (the edges) are different.

With respect to fit analysis, this dataset contains only sixteen examples, thus it is not convenient to separate between training and testing examples. Therefore, the results correspond to the whole dataset. The algorithm has been able to identify 3 different ways (with increasing level of complexity/variability) having only 16 examples. The results indicate that only one of the examples fits the first model, three examples fit the second model and three fit the third model. The fourth and most complex model, which collects all the variability of the executions, includes the rest of examples (9 out of 16).

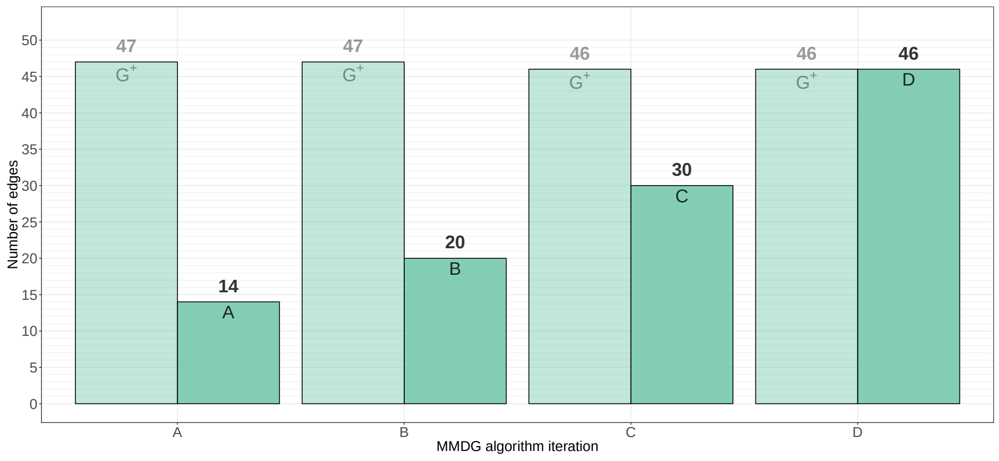


Figure 10: Brownie example. Comparison of the number of edges between the aggregate graph (pale green) and the model (dark green) that was obtained as a result for each iteration of our algorithm.

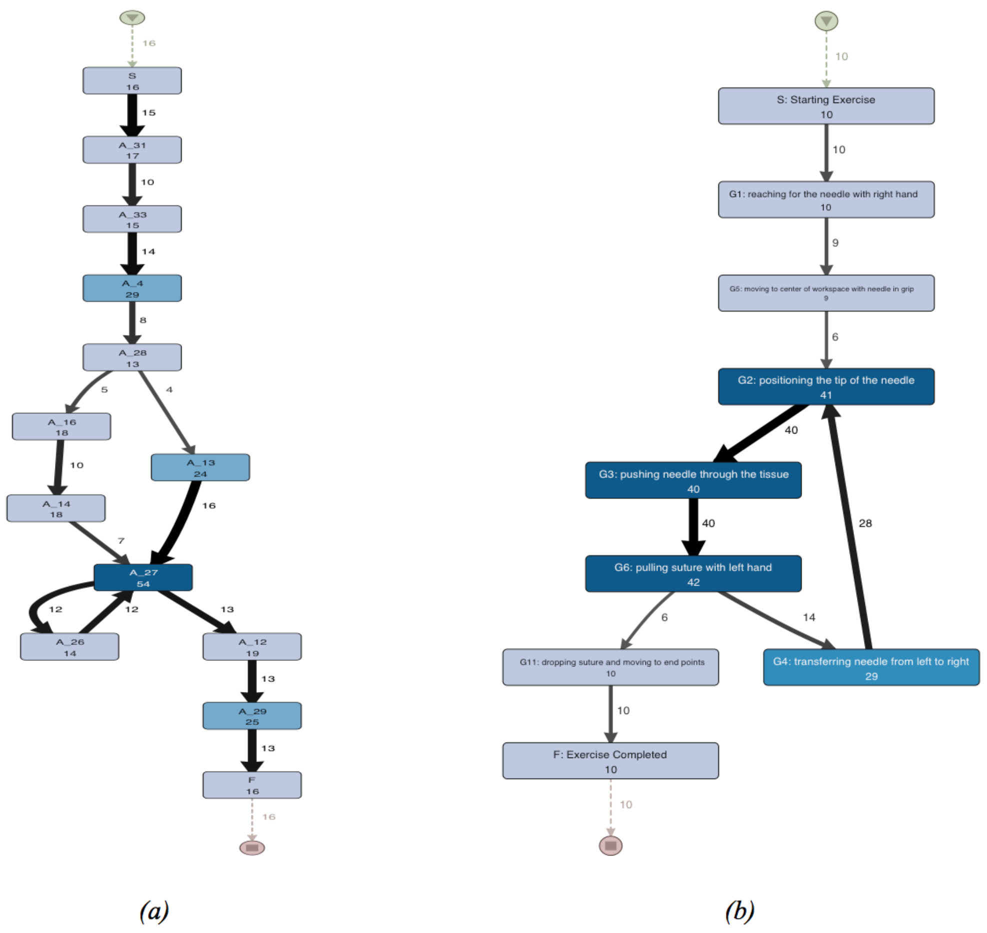
5. Discussion

Given the existence of process mining tools (van der Aalst et al., 2017), a first question is how this compare to them. In the case of learning a workflow model, the main goal of process mining is to model the process that underlies in the event logs, but not to extract a set of models with the different forms of the task and the essential activities and transitions in them. In principle, they do not assume that there may be noise, non-essential or missing activities in the process logs. Then, the whole processes are modelled. This way of doing process modelling often leads to spaghetti-like models (Günther & van der Aalst, 2007). Therefore, the commercial tools like Fluxicon DISCO[8] includes options that allow for the simplification of the obtained model based on the "popularity" of the edges. However, the final result is only one process model and it is not able to extract all the ways of performing a task. In addition, tools like DISCO do not check that the simple model obtained is supported by one example at least.

We can see some of these issues after applying DISCO to the data for the surgical and cooking domains. Figure 11 shows that DISCO only obtains one model from the aggregated graph and it is obtained by collapsing edges and nodes depending on the chosen threshold (manually determined). According to the algorithm description, the DISCO model obtained from the brownie examples has two disjoint paths in the middle of the model. In one path the user must put water (A_16) and oil (A_14) in the bowl but not the brownie bag (A_13). In the other path the user must put the brownie bag but not the water and the oil. These execution options, which are clearly wrong, were not possible with the MMDG algorithm because we check that the execution can be completed correctly following the model. Besides, we can extract not only one but all the model executions that have a minimum number of examples.

[8]https://fluxicon.com/disco/

Figure 11 also shows the model for the suture example (see Section 4.1) obtained using DISCO. In this case, the result is very similar to the model A_1 inferred from iteration 1 of MMDG applied on expert executions. However, some differences can be observed. First of all, it obtains only one model, discarding, in this way, the differences among expert executions. Second, the activity G9 does not appear, although it is executed by most of the experts in some iterations to assist in the tightening of the thread. Finally, the threshold to obtain the model must be set manually. However, this parameter is automatically set in MMDG.



S 16
A_31 17
A_33 15
A_4 29
A_28 13
A_16 18
A_13 24
A_14 18
A_27 54
A_26 14
A_12 19
A_29 25
F 16
(a)
S: Starting Exercise 10
G1: reaching for the needle with right hand 10
G5: moving to center of workspace with needle in grip 9
G2: positioning the tip of the needle 41
G3: pushing needle through the tissue 40
G6: pulling suture with left hand 42
G11: dropping suture and moving to end points 10
G4: transferring needle from left to right 29
F: Exercise Completed 10
(b)

Figure 11: Models obtained by DISCO. (a) Brownie executions model, (b) Suture executions model.

Different process mining algorithms have been also analysed, such as the Alpha-algorithm (Alves de Medeiros et al., 2004) or fuzzy models (Günther & van der Aalst, 2007). But unlike the model that can be obtained by these other methods, our ability to extract and manage several representations for the same task could be particularly beneficial for supervision. In this way, we can provide more accurate supervision to the user if we can identify their particular way of carrying out the task. To supervise a fresh example of task execution using our models, it is only necessary to replay (van der Aalst, 2016) it in any of the models obtained. Figure 12 shows a dependency graph of a fresh execution of the suture exercise. If we replay this execution in the first model obtained for the suture task, we will obtain the results shown in Figure 13. As this figure shows, by replaying the example in the model, you can detect erroneous transitions (marked in red) between the activities and those that are missing (marked in blue) and that should be carried out in order to be successful in the execution of the exercise.

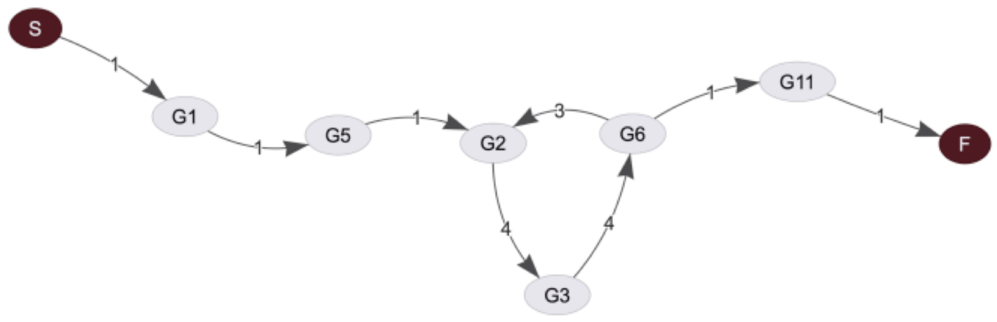


Figure 12: Fresh example of execution of the suture exercise.

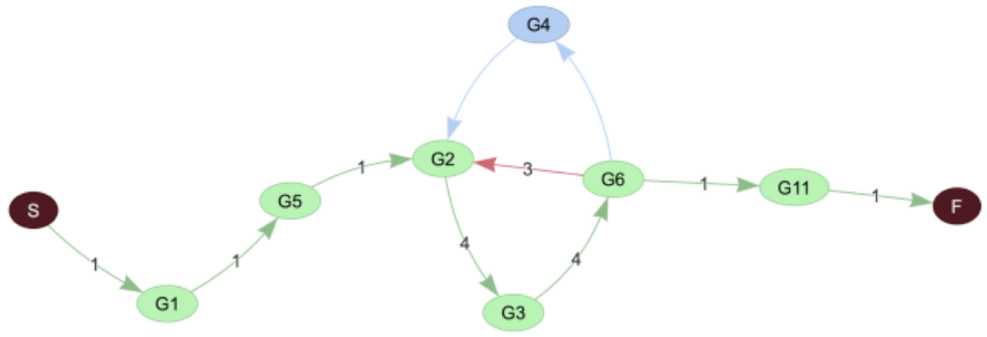


Figure 13: Result of replaying the new exercise (Figure 12) in the first model obtained from the suture examples (see Section 4.1.2). The wrong transitions are marked in red (from G6 to G2), while the missing ones are marked in blue (from G6 to G4 and from G6 to G2).

Our major aim with this paper was to build a system capable to learn the task models in contexts with very few demonstrations and having only positive examples. This is why it is required that these examples be as meaningful and rich as possible. This is supported by the results of experiments 1 and 2 set out in Section 4.1.1 and analysed in Section 4.1.2. As these experiments demonstrates, the better the GRS score of the exercises used during learning, the better the models obtained that is shown by the quality measures analysed. Thus, we have to be very careful in selecting the correct examples for the training set. The third experiment set out in Section 4.1.1 answered the question of how many examples are needed for learning the process models. In this case, we applied the algorithm to one training set that consists of the examples of the first and the second quartile based on the GRS score. What we have observed is that the resulting models do not improve the quality measures analysed in experiment 2 (that only uses examples of the first quartile). Moreover, the obtained models in experiment 3 are very similar to the ones obtained in experiment 2.

Finally, one may wonder if the use of dependency graphs could introduce a bias to the solution that limits its expressiveness. As we have mentioned, dependency graphs are an easy representation of process models that facilitate their simplification. Besides, there are tools that ease the further transformation of those models into a high-level representation such as Petri nets or BPMN (i.e., Business Process Model and Notation), or into a logic program with interesting features such as reasoning about scenarios as Event Calculus does (Cicekli & Yildirim, 2000) but that require quite more complex simplification/abstraction tools.

To close the discussion, we clarify the limitations of our approximation. First of all, we lose important information regarding the task during our mining method (e.g., the number of times the suturing cycle must be done) when we perform the aggregation of the examples. However, this meaningful information could be recovered by replaying the covered demonstrations on the corresponding obtained model. On the other hand, the semantics of an activity that is repeated several times in a sequence can vary throughout a task. This is part of representational bias that we assume by using dependency graphs. This type of representation is potentially limited to represent concurrency, duplicate actions, hierarchical activities, or model OR-splits/joins in a mined model. For instance, Yang et al. (2017) proposed a method to split duplicate activities in the final model in order to gain expressiveness. This approximation could be introduced in future versions of our algorithm. Finally, we have observed that many examples did not overlap with any model in the experiments (30% and 40% of examples). We think our criterion to determine when an example conforms to a model may be very strict. Therefore, other less restrictive conformance checking will be analysed in the future to reduce these cases.

6. Conclusions and Future Work

In this paper, we have presented a new approach to learn the different strategies for performing a task based on a few examples. Our proposal starts from an event log with the set of executions of a task in the form of sequences of activities. After converting them into dependency graphs, we apply our novel inductive method for learning models from these dependency graphs. As a result, we can generate the multiple models with the essence of the task, eliminating the noise (i.e., unnecessary transitions between activities). We have applied our approach to two challenging task: suture in minimally invasive surgery and cooking a brownie. Having in mind that we only have positive examples, we have evaluated the results with quality dimensions that are usually applied in process mining: fitness and simplicity, which is a common combination (e.g., MML/MDL principles, (Wallace & Dowe, 1999)) in situations with scarcity of data, where other regularisation terms cannot be used.

According to this balance in performance metrics, the results are quite good, showing that our approach was effective in noise reduction independently of the quality of the examples.

However, the procedure is very sensitive to the quality in the training set: increasing the quality of the examples during this phase could be translated into a significant reduction of the training set size. Thereby, if the examples are rich enough, the algorithm is able to learn good quality models with fewer examples.

In order to explore and analyse the MMDG method to other tasks provided by JIGSAWS dataset, we have developed a Shiny application[9]. On this web application, users can select a task, apply the learning models and replay the different examples (trials) on the obtained models. It also possible to watch all the steps performed by the algorithm until obtaining the final models.

In the future we will concentrate on extending our approach to also consider short descriptions in natural language of the task (performed by an expert) in order to provide the system with an automatic or semiautomatic verification step after the learning process. Finally, we will apply our developments to the automatic learning and supervision system of skilled tasks that is one of the most relevant application areas of our method.

Acknowledgements

This work has been partially supported by the EU (FEDER) and the Spanish MINECO under grants TIN2014-61716-EXP (SUPERVASION) and RTI2018-094403-B-C32, and by Generalitat Valenciana under grant PROMETEO/2019/098. David Nieves is also supported by the Spanish MINECO under FPI grant (BES-2016-078863).

References

van der Aalst, W., & Van Hee, K. M. (2004). *Workflow management: models, methods, and systems*. MIT press.

van der Aalst, W. M. (2016). *Process mining: data science in action*. Springer.

van der Aalst, W. M., Bolt, A., & van Zelst, S. J. (2017). Rapidprom: mine your processes and not just your data. *arXiv preprint arXiv:1703.03740*, .

van der Aalst, W. M., Reijers, H. A., Weijters, A. J., van Dongen, B. F., De Medeiros, A. A., Song, M., & Verbeek, H. (2007). Business process mining: An industrial application. *Information Systems*, *32*, 713–732.

Adé, H., De Raedt, L., & Bruynooghe, M. (1995). Declarative bias for specific-to-general ilp systems. *Machine Learning*, *20*, 119–154.

Agrawal, R., Gunopulos, D., & Leymann, F. (1998). Mining process models from workflow logs. In *International Conference on Extending Database Technology* (pp. 467–483). Springer.

Ahmidi, N., Tao, L., Sefati, S., Gao, Y., Lea, C., Haro, B. B., Zappella, L., Khudanpur, S., Vidal, R., & Hager, G. D. (2017). A dataset and benchmarks for segmentation and recognition of gestures in robotic surgery. *IEEE Transactions on Biomedical Engineering*, *64*, 2025–2041.

Baker, C. L., Saxe, R., & Tenenbaum, J. B. (2009). Action understanding as inverse planning. *Cognition*, *113*, 329–349.

Blum, T., Padoy, N., Feußner, H., & Navab, N. (2008). Workflow mining for visualization and analysis of surgeries. *International journal of computer assisted radiology and surgery*, *3*, 379–386.

Boger, J., Poupart, P., Hoey, J., Boutilier, C., Fernie, G., & Mihailidis, A. (2005). A decision-theoretic approach to task assistance for persons with dementia. In *IJCAI* (pp. 1293–1299). Citeseer.

[9]https://safe-tools.dsic.upv.es/shiny/SurgicalWorkflowMining/

Bouchard, B., Giroux, S., & Bouzouane, A. (2006). A smart home agent for plan recognition of cognitively-impaired patients. *JCP*, *1*, 53–62.

Buijs, J. C., Van Dongen, B. F., & van der Aalst, W. M. (2012). On the role of fitness, precision, generalization and simplicity in process discovery. In *OTM Confederated International Conferences" On the Move to Meaningful Internet Systems"* (pp. 305–322). Springer.

Camacho, R., Carreira, P., Lynce, I., & Resendes, S. (2014). An ontology-based approach to conflict resolution in home and building automation systems. *Expert Systems with Applications*, *41*, 6161–6173.

Cao, C., MacKenzie, C., & Payandeh, S. (1996). Task and motion analyses in endoscopic surgery. In *Proceedings ASME Dynamic Systems and Control Division* (pp. 583–590). Citeseer.

Carrington, P. J., Scott, J., & Wasserman, S. (2005). *Models and methods in social network analysis* volume 28. Cambridge university press.

Caruana, R. (1997). Multitask learning. *Machine learning*, *28*, 41–75.

Chen, L., Hoey, J., Nugent, C. D., Cook, D. J., & Yu, Z. (2012a). Sensor-based activity recognition. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, *42*, 790–808.

Chen, L., Nugent, C. D., Mulvenna, M., Finlay, D., Hong, X., & Poland, M. (2008). A logical framework for behaviour reasoning and assistance in a smart home. *International Journal of Assistive Robotics and Mechatronics*, *9*, 20–34.

Chen, L., Nugent, C. D., & Wang, H. (2012b). A knowledge-driven approach to activity recognition in smart homes. *IEEE Transactions on Knowledge and Data Engineering*, *24*, 961–974.

Chen, Y., Wang, J., Huang, M., & Yu, H. (2019). Cross-position activity recognition with stratified transfer learning. *Pervasive and Mobile Computing*, *57*, 1–13.

Cicekli, N. K., & Yildirim, Y. (2000). Formalizing workflows using the event calculus. In *International Conference on Database and Expert Systems Applications* (pp. 222–231). Springer.

Cook, D., Feuz, K. D., & Krishnan, N. C. (2013). Transfer learning for activity recognition: A survey. *Knowledge and information systems*, *36*, 537–556.

Crandall, A. S., & Cook, D. J. (2011). Tracking systems for multiple smart home residents. *Human Behavior Recognition Technologies: Intelligent Applications for Monitoring and Security*, (pp. 111–129).

Dai, P., Di, H., Dong, L., Tao, L., & Xu, G. (2008). Group interaction analysis in dynamic context. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, *38*, 275–282.

Ding, R., Li, X., Nie, L., Li, J., Si, X., Chu, D., Liu, G., & Zhan, D. (2019). Empirical study and improvement on deep transfer learning for human activity recognition. *Sensors*, *19*, 57.

Duan, Y., Andrychowicz, M., Stadie, B., Ho, O. J., Schneider, J., Sutskever, I., Abbeel, P., & Zaremba, W. (2017). One-shot imitation learning. In *Advances in neural information processing systems* (pp. 1087–1098).

Duong, T., Phung, D., Bui, H., & Venkatesh, S. (2009). Efficient duration and hierarchical modeling for human activity recognition. *Artificial intelligence*, *173*, 830–856.

Ericsson, K. A. (2009). *Development of professional expertise: Toward measurement of expert performance and design of optimal learning environments*. Cambridge University Press.

Finn, C., Yu, T., Zhang, T., Abbeel, P., & Levine, S. (2017). One-shot visual imitation learning via meta-learning. In *Conference on Robot Learning* (pp. 357–368).

Fürnkranz, J. (1999). Separate-and-conquer rule learning. *Artificial Intelligence Review*, *13*, 3–54.

Gao, Y., Vedula, S. S., Reiley, C. E., Ahmidi, N., Varadarajan, B., Lin, H. C., Tao, L., Zappella, L., Béjar, B., Yuh, D. D. et al. (2014). Jhu-isi gesture and skill assessment working set (jigsaws): A surgical activity dataset for human motion modeling. In *MICCAI Workshop: M2CAI* (p. 3). volume 3.

Geng, L., & Hamilton, H. J. (2006). Interestingness measures for data mining: A survey. *ACM Computing Surveys (CSUR)*, *38*, 9.

Günther, C. W., & van der Aalst, W. M. (2007). Fuzzy mining–adaptive process simplification based on multi-perspective metrics. In *International conference on business process management* (pp. 328–343). Springer.

Hoey, J., Poupart, P., von Bertoldi, A., Craig, T., Boutilier, C., & Mihailidis, A. (2010). Automated handwashing assistance for persons with dementia using video and a partially observable markov decision process. *Computer Vision and Image Understanding*, *114*, 503–519.

Hong, J.-y., Suh, E.-h., & Kim, S.-J. (2009). Context-aware systems: A literature review and classification. *Expert Systems with applications*, *36*, 8509–8522.

Hussein, A., Gaber, M. M., Elyan, E., & Jayne, C. (2017). Imitation learning: A survey of learning methods. *ACM Computing Surveys (CSUR)*, *50*, 21.

Intille, S. S., Larson, K., Tapia, E. M., Beaudin, J. S., Kaushik, P., Nawyn, J., & Rockinson, R. (2006). Using a live-in laboratory for ubiquitous computing research. In *International Conference on Pervasive Computing* (pp. 349–365). Springer.

Kalra, L., Zhao, X., Soto, A. J., & Milios, E. (2013). Detection of daily living activities using a two-stage markov model. *Journal of Ambient Intelligence and Smart Environments*, *5*, 273–285.

Kardas, K., & Cicekli, N. K. (2017). Svas: Surveillance video analysis system. *Expert Systems with Applications*, *89*, 343–361.

van Kasteren, T. L., Englebienne, G., & Kröse, B. J. (2011). Human activity recognition from wireless sensor network data: Benchmark and software. In *Activity recognition in pervasive intelligent environments* (pp. 165–186). Springer.

Kröse, B., Van Kasteren, T., Gibson, C., Van den Dool, T. et al. (2008). Care: Context awareness in residences for elderly. In *International Conference of the International Society for Gerontechnology, Pisa, Tuscany, Italy* (pp. 101–105).

Kruger, F., Nyolt, M., Yordanova, K., Hein, A., & Kirste, T. (2014). Computational state space models for activity and intention recognition. a feasibility study. *PLoS One*, *9*.

Liu, Y., Nie, L., Han, L., Zhang, L., & Rosenblum, D. S. (2015). Action2activity: recognizing complex activities from sensor data. In *Twenty-fourth international joint conference on artificial intelligence*.

Mans, R. S., Schonenberg, M., Song, M., van der Aalst, W. M., & Bakker, P. J. (2008). Application of process mining in healthcare–a case study in a dutch hospital. In *International joint conference on biomedical engineering systems and technologies* (pp. 425–438). Springer.

Alves de Medeiros, A., Van Dongen, B., van der Aalst, W., & Weijters, A. (2004). Process mining: Extending the alpha-algorithm to mine short loops. *University of Technology, Eindhoven*, *113*, 145–180.

Mueller, E. T. (2014). *Commonsense reasoning: an event calculus based approach*. Morgan Kaufmann.

Neumann, A., Elbrechter, C., Pfeiffer-Leßmann, N., Kõiva, R., Carlmeyer, B., Rüther, S., Schade, M., Ückermann, A., Wachsmuth, S., & Ritter, H. J. (2017). "kognichef": A cognitive cooking assistant. *KI-Künstliche Intelligenz*, *31*, 273–281.

Oh, J., Meneguzzi, F., & Sycara, K. (2014). Probabilistic plan recognition for proactive assistant agents. *Plan, activity, and intent recognition. Elsevier, Amsterdam, The Netherlands*, *10*, 23.

Okeyo, G., Chen, L., Wang, H., & Sterritt, R. (2011). Ontology-based learning framework for activity assistance in an adaptive smart home. In *Activity Recognition in Pervasive Intelligent Environments* (pp. 237–263). Springer.

Papadimitriou, P., Dasdan, A., & Garcia-Molina, H. (2010). Web graph similarity for anomaly detection. *Journal of Internet Services and Applications*, *1*, 19–30.

Patterson, D. J., Fox, D., Kautz, H., & Philipose, M. (2005). Fine-grained activity recognition by aggregating abstract object usage. In *Wearable Computers, 2005. Proceedings. Ninth IEEE International Symposium on* (pp. 44–51). IEEE.

Peng, L., Chen, L., Ye, Z., & Zhang, Y. (2018). Aroma: A deep multi-task learning based simple and complex human activity recognition method using wearable sensors. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, *2*, 1–16.

Raedt, L. D. (2010). Logic of generality. *Encyclopedia of machine learning*, (pp. 624–631).

Rohrbach, M., Amin, S., Andriluka, M., & Schiele, B. (2012). A database for fine grained activity detection of cooking activities. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on* (pp. 1194–1201). IEEE.

Rosen, J., Solazzo, M., Hannaford, B., & Sinanan, M. (2002). Task decomposition of laparoscopic surgery for objective evaluation of surgical residents' learning curve using hidden markov model. *Computer Aided Surgery*, *7*, 49–61.

Sadilek, A., & Kautz, H. (2012). Location-based reasoning about complex multi-agent behavior. *Journal of Artificial Intelligence Research*, *43*, 87–133.

Sánchez, D., Tentori, M., & Favela, J. (2008). Activity recognition for the smart hospital. *IEEE intelligent systems*, *23*.

Škrjanc, I., Andonovski, G., Ledezma, A., Sipele, O., Iglesias, J. A., & Sanchis, A. (2018). Evolving cloud-based system for the recognition of drivers' actions. *Expert Systems with Applications*, *99*, 231–238.

Spriggs, E. H., De La Torre, F., & Hebert, M. (2009). Temporal segmentation and activity classification from first-person sensing. In *2009 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops* (pp. 17–24). IEEE.

Stein, S., & McKenna, S. J. (2013). Combining embedded accelerometers with computer vision for recognizing food preparation activities. In *Proceedings of the 2013 ACM international joint conference on Pervasive and ubiquitous computing* (pp. 729–738). ACM.

Sukthankar, G., Geib, C., Bui, H. H., Pynadath, D., & Goldman, R. P. (2014). *Plan, activity, and intent recognition: Theory and practice*. Newnes.

Sun, X., Kashima, H., & Ueda, N. (2012). Large-scale personalized human activity recognition using online multitask learning. *IEEE Transactions on Knowledge and Data Engineering*, *25*, 2551–2563.

Twomey, N., Diethe, T., Kull, M., Song, H., Camplani, M., Hannuna, S., Fafoutis, X., Zhu, N., Woznowski, P., Flach, P. et al. (2016). The sphere challenge. *arXiv preprint arXiv:1603.00797*, .

Van Kasteren, T., Englebienne, G., & Kröse, B. J. (2010). Transferring knowledge of activity recognition across sensor networks. In *International Conference on Pervasive Computing* (pp. 283–300). Springer.

Voulodimos, A., Kosmopoulos, D., Vasileiou, G., Sardis, E., Anagnostopoulos, V., Lalos, C., Doulamis, A., & Varvarigou, T. (2012). A threefold dataset for activity and workflow recognition in complex industrial environments. *IEEE MultiMedia*, (pp. 42–52).

Wallace, C. S., & Dowe, D. L. (1999). Minimum message length and kolmogorov complexity. *The Computer Journal*, *42*, 270–283.

Xu, D., Nair, S., Zhu, Y., Gao, J., Garg, A., Fei-Fei, L., & Savarese, S. (2018). Neural task programming: Learning to generalize across hierarchical tasks. In *2018 IEEE International Conference on Robotics and Automation (ICRA)* (pp. 1–8). IEEE.

Yan, X., Zhou, X., & Han, J. (2005). Mining closed relational graphs with connectivity constraints. In *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining* (pp. 324–333). ACM.

Yang, S., Zhou, M., Chen, S., Dong, X., Ahmed, O., Burd, R. S., & Marsic, I. (2017). Medical workflow modeling using alignment-guided state-splitting hmm. In *Healthcare Informatics (ICHI), 2017 IEEE International Conference on* (pp. 144–153). IEEE.

Ye, J., Stevenson, G., & Dobson, S. (2015). Usmart: An unsupervised semantic mining activity recognition technique. *ACM Transactions on Interactive Intelligent Systems (TiiS)*, *4*, 16.

Yordanova, K., Whitehouse, S., Paiement, A., Mirmehdi, M., Kirste, T., & Craddock, I. (2017). What's cooking and why? behaviour recognition during unscripted cooking tasks for health monitoring. In *Pervasive Computing and Communications Workshops (PerCom Workshops), 2017 IEEE International Conference on* (pp. 18–21). IEEE.