Vector Based Modelling of Business Processes

Virginia Niculescu[a] and Maria-Camelia Chisăliţă-Creţu[b], Cristina-Claudia Osman[c] and Adrian Sterca[d]
Babeş-Bolyai University, Cluj-Napoca, Romania

Keywords: Business Process Formalization, Robotic Process Automation, Execution Optimization.

Abstract: Robotic Process Automation (RPA) platforms target the automation of repetitive tasks belonging to business processes, performed by human users. We are trying to increase the level of abstraction in representing complex processes (made from several conceptual operations) for RPA, by using vectors that allow not only a simple and condensed modelling, but also an efficient way towards obtaining an optimal execution order for them. Vector-based representation of the processes can serve to optimize the user-specified execution order of the conceptual operations that constitute a process. For this, we propose an optimization strategy based on a heuristic that helps us to rearrange the conceptual operations efficiently, thus reducing the total execution time of the process.

1 INTRODUCTION

Robotic Process Automation (RPA) is generally defined as the application of specific methodologies and technologies that aim to automate repetitive tasks achieved usually by human users (Institute for Robotic Process Automation, 2015), (Hofmann et al., 2020). Current RPA platforms allow an increase in work efficiency and accuracy by automatizing business processes and executing them in a more robust way by avoiding possible human errors. RPA refers to those tools that operate on the user interface (UI) aiming to perform automation tasks using an ”outside-in” approach. The information systems are kept unchanged, compared to the traditional workflow technology, which allows the improvement using an ”inside-out” approach (Van-der Aalst et al., 2018). RPA frameworks (e.g., UiPath, Automation Anywhere, Blue Prism, Microsoft Power Automate, etc.) operate (i.e. create automated business processes) in the following way: RPA developers identify UI components of a software application like buttons, text input controls, drop-down lists, and tables, and then customize activities by writing code snippets in a programming language to act on these UI controls (e.g. click the selected button, writing data in the text input, select all data from a table or a drop-down list, etc.); this code that references the selected UI controls forms the automated business process which can be executed many times later with different input parameters.

The original contributions presented in the paper can be summarized as follows: (i) a new vector-based representation of processes using building blocks; (ii) an optimization heuristic for executing business processes represented in vector spaces.

The rest of the paper is structured as follows: Section 2 provides related work about vector-based representation approaches for business processes. Vector representation of the processes is introduced in Section 3, while Section 4 describes the process execution optimization practical aspects together with the optimization heuristic designed to enhance execution efficiency. The paper concludes with the final remarks.

2 RELATED WORK

There are various definitions of business processes, most of them focusing on the flow of data and use of information and resources (OMG, 2013), (Harmon, 2016). Processes can be modeled using graphical representations like standardized or non-standardized modelling languages. The most common modelling

[a] https://orcid.org/0000-0002-9981-0139
[b] https://orcid.org/0000-0002-1414-0202
[c] https://orcid.org/0000-0002-9706-2915
[d] https://orcid.org/0000-0002-5911-0269

Niculescu, V., Chisăliţă-Creţu, M., Osman, C. and Sterca, A.
Vector Based Modelling of Business Processes.
DOI: 10.5220/0012739100003687
Paper published under CC license (CC BY-NC-ND 4.0)
In *Proceedings of the 19th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE 2024)*, pages 735-742
ISBN: 978-989-758-696-5; ISSN: 2184-4895
Proceedings Copyright © 2024 by SCITEPRESS – Science and Technology Publications, Lda.

languages used in process representations are Event-driven Process Chains (EPCs) (Keller et al., 1992) and BPMN (Business Process Model Notation) (OMG, 2013) diagrams. On the other hand, a mathematical modelling language suitable for representing processes is Petri Nets (Petri, 1962).

The utilization of vectors for representing and analyzing business processes has been extensively discussed in the literature, particularly to calculate similarity. (Jung et al., 2009) convert business process models into vector representations, considering both activities and transitions. Their objective is to assess the similarity of process models represented as vectors by using the Cosine metric. Vector spaces can also be employed to aggregate activities using the K-means clustering algorithm (Smirnov et al., 2011). In (Smirnov et al., 2011) there are two vector spaces, the first one consists of the properties associated with the activities (heterogeneous vector space). The second vector space refers to the dimensions that correspond to the property values of a particular type of activity (homogeneous vector space). (Dijkman et al., 2011) define three metrics for determining the similarity between business process models: node similarity, structural similarity, intended behavior similarity. The vector space document is defined in (Salton et al., 1975) as a collection of process models (documents), together with a set of index terms for indexing the documents and an index vector for each document. These metrics have been implemented in the ProM Framework (Van Dongen et al., 2005).

The data perspective of a process model is also important. Therefore, data-aware methods for measuring business process similarity are also analyzed (Yu et al., 2013), and(Amiri and Koupaee, 2017). An extension of Petri Nets in the context of e-commerce - EBPN (E-commerce Business Process Net) is proposed by (Yu et al., 2013). The similarity of activities is measured by considering their access to data (Amiri and Koupaee, 2017).

Business process variants refer to different versions or iterations of a business process that can vary based on various criteria (Taymouri et al., 2021).

3 VECTOR REPRESENTATION

In this paper, the term *concept* refers to the data stored in a database table, while *entity* refers to a row/record of a database table. A *concept* always describes a set of *entities*. In a previous work (Sterca et al., 2023) we proposed a web automation tool that automatically translates human user operations on the UI of a target web application onto conceptual operations in the underlying database (e.g. adding a new account to the database, updating a contact entity). This web automation tool takes the form of a browser plugin. More specifically, the human user guides the automation tool (by navigating in the target business web application) so that the plugin discovers what we call *primary blocks for process automation* - which is another name for a conceptual operation. After the plugin discovers *primary blocks*, it can also execute automatically such *primary blocks* on the target web application or it can form complex processes with the discovered *primary blocks* and it can execute those too.

We propose in this paper an approach for modelling complex processes based on defining an operation vector for each concept and considering all these vectors together to characterize a process made from several conceptual operations (i.e. primary blocks) across different concepts. In the remaining of the paper, but mostly in the present section, we will use the terms *building block* and *conceptual operation* interchangeably.

3.1 Base Vectors for Building Blocks

The building blocks discovered by the plugin tool represent all possible conceptual operations executed by the software robot. They are, as their name implies, all the ingredients the robot can use when executing automatic processes. To inform the robot regarding the operations that should be executed, we need to have a model of representation for the set of all these building blocks in a way simple enough to be interpreted by the robot and, at the same time, flexible to possible optimizations.

For example, let's assume that for a concept *Account*, the following five building blocks have been discovered: *SellectAll, Insert, Update, Delete* and *Deactivate*.

We consider a predefined order between these: *SelectAll*, *Insert*, *Update*, *Delete*, and *Deactivate*.

Consequently, for the representation of the building blocks working with the concept *Account*, we propose using the following base vectors:

- $e_1 = (1,0,0,0,0)$ – SelectAll
- $e_2 = (0,1,0,0,0)$ – Insert
- $e_3 = (0,0,1,0,0)$ – Update
- $e_4 = (0,0,0,1,0)$ – Delete
- $e_5 = (0,0,0,0,1)$ – Deactivate

Similar to the building blocks representation for the *Account* concept, we can define base vectors for all the concepts managed by an application. Still, the number of building blocks – *b*– may vary from one concept to another or from one application to another.

3.2 Abstract Process Representation

Considering a real application that works with the concepts $C_1, C_2, \ldots C_k$, the modelling of a corresponding process could be done with one vector having the size equal to $b * k$, where b is the number of all possible building blocks for each concept and k is the number of concepts handled in the application.

A series of conceptual operations (i.e. building blocks) upon a concept C can be represented as a vector :

$$P^C = (n_1^C \cdot e_1^C \quad n_2^C \cdot e_2^C \quad n_3^C \cdot e_3^C \quad n_4^C \cdot e_4^C \quad n_5^C \cdot e_5^C)$$

where n_1^C are the number of *SellectAll* operations upon C, n_2^C are the number of *Insert* operations upon C, etc.

If we consider a predefined order between the concepts, the operations on all the concepts could be represented as:

$$P = ()^{C_1} +\!\!+ ()^{C_2} +\!\!+ ()^{C3} \ldots.$$

where the operator $+\!\!+$ is the concatenation operator.

This modelling is possible if there is a predefined order established between all the concepts, and we consider that for all concepts we have the same number b of possible operations applied to them. The predefined order does not specify implicit constraints between the concepts - it is defined just to have a structured way of modelling.

A general process could be represented as one vector of size $b * k$, (e.g., $b = 5,\ k = 5$), obtained through the concatenation of the vectors corresponding to the operations identified for each concept.

$$P = \quad \bigoplus_{i=1}^{k} (n_1{}^{C_i} \cdot e_1^{C_i} +\!\!+ n_2^{C_i} \cdot e_2^{C_i} +\!\!+ n_3^{C_i} \cdot e_3^{C_i} +\!\!+ n_4^{C_i} \cdot e_4^{C_i} +\!\!+ n_5^{C_i} \cdot e_5^{C_i})$$

where $\bigoplus$ is the concatenation operator.

The type of operations that are specified in a process could be represented using a vector O that expresses only the fact that one type of operation is or is not included. This could be defined as:

$$O = \quad \bigoplus_{i=1}^{k} (x_1{}^{C_i} \cdot e_1^{C_i} +\!\!+ x_2^{C_i} \cdot e_2^{C_i} +\!\!+ x_3^{C_i} \cdot e_3^{C_i} +\!\!+ x_4^{C_i} \cdot e_4^{C_i} +\!\!+ x_5^{C_i} \cdot e_5^{C_i})$$

where x_i could be either 0 or 1.

In addition, the number of operations $n_1, n_2, \ldots$ may be aggregated in a vector M, as follows:

$$M = \quad \bigoplus_{i=1}^{k} (n_1^{C_i} +\!\!+ n_2^{C_i} +\!\!+ n_3^{C_i} +\!\!+ n_4^{C_i} +\!\!+ n_5^{C_i})$$

Therefore, the process representation is a pair of vectors (O, M) and the process is

$$P = O \odot M,$$

where $\odot$ is the element-wise product (Hadamard product (Horn, 1990)) of the operations vector O and the multiplicity vector M.

Remarks:

- If the number of possible operations differs from one concept to another the following two options could be applied:
  - to consider b as being the maximum number of operations possible to be applied on any concept; based on the concepts' order the start index for each concept could be identified ($modulo\ b$ operation).
  - to define a configuration that specifies the number of operations on each concept together with their significance; this configuration will allow computing the start index corresponding to each concept.
- The concrete parameters of each operation are considered to be part of the *data plan* and will be specified separately by the abstract vector representation of the process in order not to overload the abstract representation. For example, for the operations *Insert* of the concept *Account* ($e2_{Account}$), the data plan should contain parameters in the following form:

  $e2_{Account}$:
  $AccountName$: *"CompanyA"*, $Email$: *"john@.."*, ...,
  $AccountName$: *"CompanyB"*, $Email$: *"jane@.."*, ...,
  ...
- The relations between the concepts emphasized by the relations between the corresponding tables of the database may emphasize an order between the operations on different concepts. These relations are static – they are not supposed to be changed during the application usage. Still, since these dependencies could be cyclic, we cannot consider that the operations may be executed in the order they appear in the vector representation. The execution order should be primarily based on the order given by the user.
- A complete representation of a process should include besides the vector P, the associated data plan that contains all the corresponding data parameters, and an execution order.

3.3 Example of a Process Representation

To illustrate this modelling we will consider an example with the following concepts *Account, Contact, Lead, Appointment, Opportunity*, which are the 5 concepts identified in Microsoft Dynamics 2016 CRM [1].

[1]https://www.microsoft.com/en-us/dynamics-365

The order between the concepts is that given by the previous enumeration.

Since in this case, we will have 5 possible building blocks for each concept, as a result, the vectors that represent the processes will have a size equal to (5 concepts x 5 operations/concept).

A particular process that we consider may consist of the following conceptual operations:

Listing 1:

1. (Insert Contact C1)
2. (Update Lead L1)
3. (Insert Account A1)
4. (Update Account A1) - using Lead L1
5. (Insert Contact C2)
6. (Update Account A2) - using Contact C2
7. (Delete Appointment X1)
8. (Insert Account A3)
9. (Delete Opportunity O1)
10. (Deactivate Opportunity O2)
11. (Delete Appointment X2)
12. (Insert Opportunity O3)
13. (Update Account A1) - using O3
14. (Update Lead L2)
15. (Delete Account A4)

We assume that the order of the operations provided by the user is valid, i.e., update, delete, and deactivate operations that are used for the entities available in the database may be successfully executed. The vector-based representation of the above process, where the concepts are ordered as *Account*, *Contact*, *Lead*, *Appointment*, and *Opportunity*, is the following:

P = (0,2,3,1,0; 0,2,0,0,0; 0,0,2,0,0; 0,0,0,2,0; 0,1,0,1,1)

The corresponding *O* and *M* vectors are:

O = (0,1,1,1,0; 0,1,0,0,0; 0,0,1,0,0; 0,0,0,1,0; 0,1,0,1,1)

M = (0,2,3,1,0; 0,2,0,0,0; 0,0,2,0,0; 0,0,0,2,0; 0,1,0,1,1)

The execution order as specified by the user is

```
E = (  [],[3,8],[4,6,13],[15],[];
       [],[1,5],[],[],[];
       [],[],[2,14],[],[];
       [],[],[],[7,11],[];
       [],[12],[],[9],[10] )
```

where for each type of operation is provided a list of indices that states the order of each concrete operation in the process specified by the user. An empty list [] suggests that the corresponding operation is not part of the process.

3.4 Usefulness of the Vector Representation

Representing complex automated processes as vectors has several benefits. The first one would be that we can easily compare different processes. In our previous work that describes the browser plugin used for web automation, we represented complex processes as JSON (Javascript Object Notation) expressions which included both the conceptual operations and their arguments (i.e the data plan). Using such representation it is difficult to decide if two complex processes (made from tens of building blocks) are equal (i.e. they have the same building blocks, but maybe in a different order) or one of them is a prefix/suffix of another or they both have a common subpart. By using the representation of a process as a *P* vector and dissociating the data plan, these decisions are more readily made. Such comparisons between processes are useful in process mining.

Representing a complex process as a vector can also be helpful when trying to optimize the executions of the conceptual operations that make a complex process. The next section discusses the possible execution optimizations and introduces an optimization heuristic for improving the process execution efficiency based on the vector representation previously discussed.

4 PROCESS EXECUTION OPTIMISATION

Complex business processes emphasize the existence of multiple similar operations that, most of the time, are spread over the entire process. Optimizing the execution of operations within a process can be achieved based on the paths chosen by the user at the UI level.

4.1 Concrete Example

We consider the following complex process that performs operations on *Account* and *Contact* concepts available in the Microsoft Dynamics 2016 CRM application. The access paths to operations, starting from the root node, are:

- (BurgerMenu→Sales→Contacts→New C1→Save),
- (BurgerMenu→Sales→Accounts →New A1→Save),
- (BurgerMenu→Sales→Accounts→UpdateA1→Save),
- (BurgerMenu→Sales→Contacts→New C2→Save),
- (BurgerMenu→Sales→Accounts→New A2→Save),
- (BurgerMenu→Sales→Accounts→Update A2→Save)

BurgerMenu refers to a global menu of the application, *Sales* is a submenu of the *BurgerMenu* and *Contacts* is a submenu item of submenu *Sales*. *Insert C1* is the UI form that allows a new Contact C1 to be added to the underlying database (this UI form is triggered by the user when clicking on a *New*-like button). *Save* is just the *Save* button of the application that triggers the saving of the currently added Contact entity. Similarly, there are other examples of navigation paths in the above example: a navigation path for adding a new Account, A1 and respectively A2, and a navigation path for updating an existing Account. For automatic execution, each line represents one building block. For example, when executing the first building block, the automation tool must click on the *BurgerMenu*, then it must click on the *Sales* submenu, then it must click on the *Contacts* submenu, then it must trigger a click on the button that allows the user to add a new Contact and after filling all the required fields, it will finally click on the *Save* button.

A more efficient way to execute the specified process is to group the operations of the same type:

- (BurgerMenu→Sales→Contacts→
  - New C1→Save
  - New C2→Save)
- (BurgerMenu→Sales→Accounts→
  - New A1→Save
  - New A2→Save

  - Update A1→Save
  - Update A2→Save)

The optimization of the process execution reflects the possibility of avoiding several *BurgerMenu*-based actions that occur in a full path operation execution. This results in a reduced number of interactions with the UI elements that increase efficiency and decrease the execution time.

4.2 Optimisation Heuristic

Based on the vector process representation specified in section 3 we propose a strategy for obtaining an optimized execution.

The optimization strategy is based on a heuristic elaborated on the following observations: (i) clustering the operations of identical types on the same concept improves the efficiency of the execution (by eliminating the need for full path menu navigation); (ii) *Insert* operations should be executed first (they should not be postponed); (iii) *Update, Delete* and *Deactivate* operations could be postponed - they are constrained only by the previous specific *Insert* operations.

Following the vector modelling of the process we will use similar vectors to compute the clusters (group of operations that should be executed at the same stage). In this case, we will store into the vector, not the number of operations but their order in the execution.

Starting from the order given by the user, the following mathematical rules should be applied when clusters are created:

- if two or more *Insert* operations on the same concept appear in the user list, then the *minimum position* should be considered for them;
- if two or more *Update, Delete* or *Deactivate* operations on the same concept appear in the user list, then the *maximum position* should be considered for them.

For the business process provided in Subsection 3.3 we define a vector that helps us to rearrange these primary block operations to optimize the execution on the interface. The execution vector is similar to that used for modelling a process (see Subsection 3.1), but it contains the execution order of the operations, instead of the number of operations of the same type included in the process.

The application of the proposed heuristic points out three stages. The first stage corresponds to **creating clusters** of similar operations on the same concept that use different entities.

The first cluster consists of operations *(Insert Contact C1)* and *(Insert Contact C2)* that had the initial execution orders 1 and 5. They are placed in the same group, with the intent to execute based on the *min* rule, defined for the *Insert* operation, i.e., $min(1,5) = 1$.

For operations *(Update Account A1) - using Lead L1* and *(Update Account A2) - using Contact C2*, a cluster is created considering the *max* rule for the Update operation, i.e., $max(4,6) = 6$, where 4 and 6 were in the initial order of these *Update* operations.

The second stage is represented by the application of the rule that allows to **group clusters** for *Update*, *Delete*, and *Deactivate* operations that work on the same concept and postpone it. Therefore, the concept *Contact* emphasizes clusters 13 and 15 for the *Update* and *Delete* operations. The *max* rule is applied to them, i.e., $max(13,15) = 15$ meaning that *Update* operations from cluster 13 will be postponed and executed together with cluster 15.

The last stage allows to **normalize the cluster identifiers** in the execution vector, by providing new cluster identifiers in ascending order with no gaps between subsequent clusters. The stage is required as the previous phases created clusters or grouped multiple clusters to the larger identifier, causing gaps in cluster numbering.

Since the operations *Update, Delete* and *Deactivate* are all commutative, we may try to cluster these operations on each concept. To do this, for each concept, we change the values corresponding to these operations with the maximum value of them. The 0 values are not modified since they emphasize that there is no such operation.

From this vector, we can obtain a better order of

Listing 3: Step 1 – (Insert Contact C1)

0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Step 2 – (Update Lead L1)

0	0	0	0	0	0	1	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0

Step 3 – (Insert Account A1)

0	3	0	0	0	0	1	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0

Step 4 – (Update Account A1) - using Lead L1

0	3	4	0	0	0	1	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0

Step 5 – (Insert Contact C2) => min(1,5)=1

0	3	4	0	0	0	1	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0

Step 6 – (Update Account A2) - using Contact C2 => max(4,6)=6

0	3	6	0	0	0	1	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0

Step 7 – (Delete Appointment X1)

0	3	6	0	0	0	1	0	0	0	0	0	2	0	0	0	0	0	7	0	0	0	0	0	0

Step 8 – (Insert Account A3) => min(3,8)=3

0	3	6	0	0	0	1	0	0	0	0	0	2	0	0	0	0	0	7	0	0	0	0	0	0

Step 9 – (Delete Opportunity O1)

0	3	6	0	0	0	1	0	0	0	0	0	2	0	0	0	0	0	7	0	0	0	0	9	0

Step 10 – (Deactivate Opportunity O2)

0	3	6	0	0	0	1	0	0	0	0	0	2	0	0	0	0	0	7	0	0	0	0	9	10

Step 11 – (Delete Appointment X2) => max(7,11)=11

0	3	6	0	0	0	1	0	0	0	0	0	2	0	0	0	0	0	11	0	0	0	0	9	10

Step 12 – (Insert Opportunity O3)

0	3	6	0	0	0	1	0	0	0	0	0	2	0	0	0	0	0	11	0	0	12	0	9	10

Step 13 – (Update Account A1) - using O3 => max(6,13)=13

0	3	13	0	0	0	1	0	0	0	0	0	2	0	0	0	0	0	11	0	0	12	0	9	10

Step 14 – (Update Lead L2) => max(2,14)=14

0	3	13	0	0	0	1	0	0	0	0	0	14	0	0	0	0	0	11	0	0	12	0	9	10

Step 15 – (Delete Account A4)

0	3	13	15	0	0	1	0	0	0	0	0	14	0	0	0	0	0	11	0	0	12	0	9	10

Step 16 – Grouping Update, Delete, and Deactivate

0	3	15	15	0	0	1	0	0	0	0	0	14	0	0	0	0	0	11	0	0	12	0	10	10

Step 17 – Normalization

0	2	7	7	0	0	1	0	0	0	0	0	6	0	0	0	0	0	4	0	0	5	0	3	3

execution. For each operation of a certain type, we obtain the number of the cluster by taking the corresponding value from the vector. Since it is based on applying a heuristic, this order is not optimal from an efficiency point of view, but it increases the efficiency by grouping certain operations applied to the same concept.

Therefore, the execution order of the operations from the initial process is the following:

Listing 2:

1. (Insert Contact C1)
1. (Insert Contact C2)
2. (Insert Account A1)
2. (Insert Account A3)
3. (Delete Opportunity O1)
3. (Deactivate Opportunity O2)
4. (Delete Appointment X1)
4. (Delete Appointment X2)
5. (Insert Opportunity O3)
6. (Update Lead L1)
6. (Update Lead L2)
7. (Update Account A1) - using Lead L1
7. (Update Account A2) - using Contact C2
7. (Update Account A1) - using O3
7. (Delete Account A4)

The execution order vector specifies the cluster for each operation specified by the process vector described in Section 3, i.e. the corresponding index in the execution vector represents the cluster number of the operations specified by the process vector.

$$P = (0,2,3,1,0;\ 0,2,0,0,0;\ 0,0,2,0,0;\ 0,0,0,2,0;\ 0,1,0,1,1)$$
$$E_h = (0,2,7,7,0;\ 0,1,0,0,0;\ 0,0,6,0,0;\ 0,0,0,4,0;\ 0,5,0,3,3)$$

For example, the value on the second index in the vector P specifies two *Insert* operations on concept *Account* which, based on the second value in vector E

will be executed in cluster 2. The order between the operations in one cluster is given by the initial user order.

Compared to the initial execution vector E, the optimized execution vector E_h obtained after the heuristic application will not have associated a list of indices for every concept operation, but a single value, i.e., a cluster identifier. This indicates a simplified and improved variant of the initial execution vector.

For instance, in the execution vector E for the *Account* concept there are the lists of operations $[4,6,13]$ and $[15]$ that are associated with the *Update* and *Delete* conceptual operations. This involves the execution of operation 4 - *(Update Account A1) - using Lead L1*, followed by operation 5 - *(Insert Contact C2)*, meaning that the user needs to switch from concept *Account* to concept *Contact*.

The optimisation heuristic applied step-by-step on the previous process example is available in **Listing 3**. By grouping similar operations on the same concept that use different entities in the same cluster, e.g., *(Insert Contact C1)* and *(Insert Contact C2)* in cluster 1, they are executed together, *reducing the number of interactions* with the user interface components, e,g, buttons, and the *lowering the execution time*.

The **validation rule** that ensures the fact that each operation is included in an execution cluster and there are no empty clusters is:
For every non-zero index of the process representation vector, there is a non-zero index that corresponds to an identified cluster in the process execution vector.

4.3 Numerical Evaluation

We can evaluate the time reduction obtained when executing a complex process like the one depicted in **Listing 1** from section 3.3 using the optimization heuristic execution described in sections 4.2, by considering what happens when our automation browser plugin executes a single building block. Even if, the evaluation in this section is applied to a concrete example from Microsoft Dynamics 2016 CRM, the results should be similar to other business web applications although the actual time reduction value can be different.

When executing the first building block of the process depicted in Listing 1, our automation browser plugin must click on the *BurgerMenu*, then it must click on the *Sales* submenu, then it must click on the *Contacts* submenu, then it must trigger a click on the button that allows the user to add a new *Contact* and after filling all the required fields, it must finally click on the *Save* button. After triggering each click event, the plugin must wait a time interval so that this click event is handled by the web application. The handling of a click event can consist of a complete document reload or one or several XHR requests accompanied by DOM (Document Model Object) updates in the current document. But our automation browser plugin has no way of knowing how much time to wait for the completion of the handling of the click event. So it always waits for a specific time interval, t_{delay}, after it triggers a UI event (e.g. a click). For Microsoft Dynamics 2016 CRM application $t_{delay} = 5$ seconds.

If we ignore the time duration of triggering click events and filling text inputs and assume that it is 0, then the time duration of executing the *(Insert Contact C1)* building block is equal to $5 \cdot t_{delay}$ seconds. This is because clicking on the *BurgerMenu* takes t_{delay} seconds, then clicking on the *Sales* submenu takes another t_{delay} seconds, then clicking on the *Contacts* submenu takes another t_{delay} seconds, then clicking the *New* button takes another t_{delay} seconds, and finally, clicking on the *Save* button takes t_{delay} seconds.

But if the next building block executed after *(Insert Contact C1)* is *(Insert Contact C2)* as in **Listing 1** from section 5.2, then the prefix (BurgerMenu$\rightarrow$Sales$\rightarrow$Contacts) of the second building block's path is no longer required because we are operating on the same concept (so we don't have to go through all menus starting from the top-level one). In order to execute block *(Insert Contact C2)*, we just need to click on the *New* button which takes t_{delay} seconds and then click on the *Save* button which takes t_{delay} seconds. So in total, the second building block took only $2 \cdot t_{delay}$ seconds when using the heuristic.

We can compute in this way the time duration of executing the same process in the initial order (given by the user) depicted in **Listing 1** and compare it with the time duration of executing the process in the order given by our heuristic - which is depicted in **Listing 2**. This comparison is presented in Table 1, where each line represents the time duration of executing the corresponding building block in both execution orders. For example, line 2 represents the time duration when executing *(Update Lead L1)* from **Listing 1** and, respectively, the time duration when executing *(Insert Contact C2)* from **Listing 2**. Table 1 shows that when using our heuristic we obtain a time reduction of $18 \cdot t_{delay} = 18 \cdot 5 = 90$ seconds (for Microsoft Dynamics 2016 CRM $t_{delay} = 5$ seconds).

5 CONCLUSIONS

The paper introduces a formal representation of business processes that uses base vectors for the operations on the concepts managed by an application. A

Table 1: Time comparison between the user given execution order and the heuristic given execution order.

Operation no.	Listing 1 order	Listing 2 order
1	$5 \cdot t_{delay}$	$5 \cdot t_{delay}$
2	$5 \cdot t_{delay}$	$2 \cdot t_{delay}$
3	$5 \cdot t_{delay}$	$5 \cdot t_{delay}$
4	$2 \cdot t_{delay}$	$2 \cdot t_{delay}$
5	$5 \cdot t_{delay}$	$5 \cdot t_{delay}$
6	$5 \cdot t_{delay}$	$2 \cdot t_{delay}$
7	$5 \cdot t_{delay}$	$5 \cdot t_{delay}$
8	$5 \cdot t_{delay}$	$2 \cdot t_{delay}$
9	$5 \cdot t_{delay}$	$5 \cdot t_{delay}$
10	$2 \cdot t_{delay}$	$5 \cdot t_{delay}$
11	$5 \cdot t_{delay}$	$2 \cdot t_{delay}$
12	$5 \cdot t_{delay}$	$5 \cdot t_{delay}$
13	$5 \cdot t_{delay}$	$2 \cdot t_{delay}$
14	$5 \cdot t_{delay}$	$2 \cdot t_{delay}$
15	$5 \cdot t_{delay}$	$2 \cdot t_{delay}$
Total	$69 \cdot t_{delay}$	$51 \cdot t_{delay}$

process is abstractly represented by a vector formed by concatenating the vectors representing operations on each concept. Such a process vector emphasizes the types of operations included in the process and their number. This is useful when comparing two processes and when doing process mining.

The approach offers several benefits and opens new research perspectives. It provides a consistent representation of concepts, processes, dependencies, and operation multiplicity due to the vector-based representation.

Application maintenance and upgrades may be easily accommodated with our vector-based approach when new concepts, attributes, or operations are added to the application.

Additionally, we propose a heuristic for process execution optimization. The heuristic is based on grouping operational blocks that form a process into clusters that specify an optimized execution order. The clusters are specified using again a vector representation that specifies for each type of operation the cluster to which it belongs. This organization of the operations into clusters leads to a more efficient execution by shortening the navigation paths through the UI interface and thus, reducing execution time. Moreover, the identification of clusters of non-conflicting operations eases the identification of operations eligible for parallel execution.

REFERENCES

Amiri, M. J. and Koupaee, M. (2017). Data-driven business process similarity. *IET Software*, 11(6):309–318.

Dijkman, R., Dumas, M., Van Dongen, B., Käärik, R., and Mendling, J. (2011). Similarity of business process models: Metrics and evaluation. *Information Systems*, 36(2):498–516.

Harmon, P., editor (2016). *Business Process Change A Business Process Management Guide for Managers and Process Professionals, Third Edition*. Morgan Kaufmann.

Hofmann, P., Samp, C., and Urbach, N. (2020). Robotic process automation. *Electronic Markets*, 30(1):99–106.

Horn, R. A. (1990). The hadamard product. In *Proc. Symp. Appl. Math*, volume 40, pages 87–169.

Institute for Robotic Process Automation (2015). Introduction to robotic process automation. A primer.

Jung, J.-Y., Bae, J., and Liu, L. (2009). Hierarchical clustering of business process models. *International Journal of Innovative Computing, Information and Control*, 5(12):1349–4198.

Keller, G., Scheer, A.-W., and Nüttgens, M. (1992). *Semantische Prozeßmodellierung auf der Grundlage" Ereignisgesteuerter Prozeßketten (EPK)"*. Inst. für Wirtschaftsinformatik.

OMG (2013). Business Process Model and Notation (BPMN) Specification, Version 2.0.2.

Petri, C. A. (1962). Kommunikation mit Automaten. Dissertation, Schriften des IIM 2, Rheinisch-Westfälisches Institut für Instrumentelle Mathematik an der Universität Bonn, Bonn.

Salton, G., Wong, A., and Yang, C. S. (1975). A vector space model for automatic indexing. *Communications of the ACM*, 18(11):613–620.

Smirnov, S., Reijers, H. A., and Weske, M. (2011). A semantic approach for business process model abstraction. In *Advanced Information Systems Engineering: 23rd Int. Conf., CAiSE 2011, London, UK, June 20-24, 2011. Proceedings 23*, pages 497–511. Springer.

Sterca, A., Niculescu, V., Chisăliţă-Creţu, M.-C., and Osman, C.-C. (2023). Primary building blocks for web automation. In Zhang, F., Wang, H., Barhamgi, M., Chen, L., and Zhou, R., editors, *Web Information Systems Engineering – WISE 2023*, pages 376–386, Singapore. Springer Nature Singapore.

Taymouri, F., La Rosa, M., Dumas, M., and Maggi, F. M. (2021). Business process variant analysis: Survey and classification. *Knowledge-Based Systems*, 211:106557.

Van-der Aalst, W. M. P., Bichler, M., and Heinzl, A. (2018). Robotic process automation. *Business and Information Systems Engineering*, 60:269–272.

Van Dongen, B. F., de Medeiros, A. K. A., Verbeek, H., Weijters, A., and van Der Aalst, W. M. (2005). The prom framework: A new era in process mining tool support. In *Applications and Theory of Petri Nets 2005: 26th Int. Conf., ICATPN 2005, Miami, USA, June 20-25, 2005. Proceedings 26*, pages 444–454. Springer.

Yu, W., Yan, C., Ding, Z., Jiang, C., and Zhou, M. (2013). Modeling and validating e-commerce business process based on petri nets. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 44(3):327–341.