Generating Business Process Recommendations with a Population-Based Meta-Heuristic

Steven Mertens(✉), Frederik Gailly, and Geert Poels

Department of Business Informatics and Operations Management,
Faculty of Economics and Business Administration, Ghent University,
Tweekerkenstraat 2, 9000 Ghent, Belgium
{steven.mertens, frederik.gailly, geert.poels}@ugent.be

Abstract. In order to provide both guidance and flexibility to users during process execution, recommendation systems have been proposed. Existing recommendation systems mainly focus on offering recommendation according to the process optimization goals (time, cost…). In this paper we offer a new approach that primarily focuses on maximizing the flexibility during execution. This means that by following the recommendations, the user retains maximal flexibility to divert from them later on. This makes it possible to handle (possibly unknown) emerging constraints during execution. The main contribution of this paper is an algorithm that uses a declarative process model to generate a set of imperative process models that can be used to generate recommendations.

Keywords: Business processes · Recommender systems · Declarative process model · Run-time flexibility

1 Introduction

A business process is a set of one or more connected activities which collectively realize a particular business goal [1]. These processes can be formally represented by using one of numerous business process modelling languages (e.g., BPMN, UML, BPEL, Petri-net…). Business process engines (BPE) offer support for the implementation of business processes by enabling the execution based on the respective process model. They provide a software framework to handle human and non-human interaction and to ensure conformance with the specified process models.

In highly dynamic environments that need to offer high flexibility (e.g., hospitals and personalized customer service departments) it is hard to find a single fitting process model, and even more difficult to find an optimal one. There are many possible paths that should be allowed and many, often complex, restrictions to keep in mind. As example, consider the simplified case of a cancer treatment center for short stays (see Sect. 3 for more details). In this environment a set of rules (e.g., the process starts when the registration of the patient, a CT or MRI scan has to be directly followed by a doctor's visit…) and limitations (e.g., taking a strong painkiller means that a chemotherapy session is no longer possible in the remainder of the process instance) apply. The order of the activities for a certain process instance cannot be predetermined; instead it has to be personalized according to the additional emergent constraints (e.g., the patient is allergic to the painkiller) of the specific instance. For a doctor it can be difficult to correctly apply all these rules (although in the simplified example this is manageable) on specific variations of the process that possibly have been handled by other doctors up to that point.

The resulting question is: how can business process engines offer support and flexible assistance in this context? A differentiation has to be made between build-time flexibility and run-time flexibility. The first is intrinsic to the created model, the latter is the flexibility allowed by a process after being deployed [2].

An approach to add build-time flexibility to the process would be to create a model that entails all possible paths. A first issue arising here is that the model would be very big and complex making human understanding difficult. This is, however, not as much of a problem when using business process engines, since human understanding isn't strictly necessary once the model is deployed. In addition, the size of the model can also make the implementation of all paths significantly more time consuming. In static environments this is not that important, but this step would have to be repeated on regular basis in dynamic environments where the constraints change frequently. Finally, a problem concerning the decisions emerges when increasing the number of paths represented in the process model. It is one thing to allow many paths, another to choose one for a specific instance of the process. This responsibility is shifted to the process participants, because no information is offered on the advantages and/or disadvantages of choosing a certain option.

As stated in [3], there is an apparent paradox between providing guidance and flexibility in how to proceed during process execution. Guidance is often thought of as forcing the user in a certain direction. This can be countered by using a business process engine offering recommendations to the users. A recommendation entails a single activity or a set of parallel activities to be executed next based on a certain goal, considering previously executed activities (i.e., a process instance that was started but not completed yet) [4]. They are ordered based on criteria not necessarily visible to the user. The user is encouraged to choose the 'best' recommendation, but is free to choose another one based on the specific circumstances that apply (e.g., patient requests an additional CT-scan before consenting with an operation). In the cancer treatment center, it would be beneficial to have a system that makes these recommendations based on the details of the specific case, while making sure that the next step is also conformant with the process in general. The approach is similar to what a GPS-system offers users compared to a street map. The GPS-system shows the user step by step how to navigate to his destination. If the user chooses to diverge from the recommended path, the GPS-system will offer an alternative optimized path based on its current position, and thus based on the previous choices of the user. In the ideal situation it also will not propose paths that are not possible (e.g., wrong way in a one-directional street). This alleviates some of the responsibilities placed on the user, while offering flexibility in the form of run-time flexibility. Similar recommendation systems have already been proposed in [3–5]. In [3, 4] process mining techniques are used to calculate or estimate the criteria (e.g., shortest duration, lowest cost…) to sort the recommendations.

The approach of [5] uses constraint programming techniques to produce business enactment plans, which are used to generate recommendations.

A common starting point for the systems in [3] and [5] is a declarative process model (e.g., a Declare model [6]) as opposed to a specific imperative process model (e.g., BPMN), just like a GPS-system internally uses a map combined with additional information (e.g., on-way streets, traffic information…) and not one path. Declarative process models [6] comprise a specification of the environment, its limitations and its rules in terms of a set of constraints. This gives leeway to follow different paths and avoids over-specification. When changes to the environment occur, the set of constraints is all that needs to be adjusted in the system itself. Imperative process models on the other hand, represent the precise (often overspecified) control-flow of a business process. They can still be used internally to generate possible valid paths based on the declarative model, but they are not necessarily visible to users of the system (only recommendations for the next step are visible).

A limitation of all three mentioned recommendation systems is that they only focus on the direct optimization goals of the process. The systems are most suitable for processes that require limited flexibility and variability. They generally assume the top recommendation will be followed by the user. It is, however, possible that the choice to initially follow an optimal path limits the freedom to diverge to other paths further down the road. So in the context of a highly dynamic environment with high flexibility needs, these systems will not produce the results we are looking for. The path they are suggesting might be optimal at the time it is generated, but when the context changes during the execution of the activities in that path, it might lose its optimal position. Therefore, specifically for highly dynamic environments, we believe a technique to generate robust process models (i.e., in a sense immune against bounded uncertainty) is still missing.

The main contribution of this paper is thus to propose an alternative to the aforementioned systems: a robust process engine that can deal with changes to its very dynamic and complex environment at run-time. Hence, it has to try and find a balance between the optimization goals of the process according to the current context and maximizing the freedom to diverge from the recommended paths (e.g., because of emergent constraints or requirements). We will primarily focus on the latter, offering recommendations that allow the most flexibility in the later steps of the process. Also, this indirectly entails checking the feasibility of possible next actions.

The novel idea is to use a population-based meta-heuristic for the generation of a set of imperative process models based on a declarative model (e.g., Declare). In the context of a very dynamic and complex environment that requires high flexibility, this offers some interesting advantages. The most important advantage is that the technique allows us to incorporate a new measure for run-time flexibility based on the population kept. The measure provides an estimate of the run-time flexibility of a certain imperative process model relative to the set of models in the population.

The remainder of this paper will first discuss the idea behind the proposed algorithm as well as the actual implementation algorithm in Sect. 2. This is followed by a short demonstration using the example of the cancer treatment center in Sect. 3. In Sect. 4 we discuss an important optimization. Finally, a summary of the contribution of this paper and further research will be presented in Sect. 5.

2 Solution Design

The developed population-based meta-heuristic takes its inspiration from nature and more specific from the artificial immune system (AIS) [7], which in turn is based on the vertebrate immune system. It is tasked with the protection of the organism from malfunctioning cells in the body (e.g., cancers) and foreign diseases-causing elements, called antigens. Two major groups of immune cells, called B-cells and T-cells, are tasked with identifying and stopping antigens from going rampant through the body of the organism. While being generally similar, they differ in how they recognize antigens. The algorithm, described below, is based on the functioning of the B-cells.

The environment in which the AIS functions is very similar to the problem environment described in the introduction of this paper. The solution space is also very large and dynamic, which makes finding an optimal solution hard and one that stays optimal in a changing environment impossible. This makes the AIS approach a great fit for calculating recommendations in a very dynamic and complex environment that requires high flexibility. The fact that the solution could be a suboptimum is not a big problem in this context, as the focus is on providing a robust solution.

Figure 1 gives an overview of the proposed implementation. At the start a set of random unique imperative process models, called the random population, is generated. The only requirement for these models is that they are valid according to the set of constraints contained by the declarative model. The initial population is created with the best models (based on the fitness function discussed below) from the random population. This initial population is sorted (again based on the fitness function) and will then be used as input for the iterative steps of the algorithm: clonal selection, mutation and selection. The three iterative steps, described below, will be executed until the stop criteria are met (i.e., 100 iterations done or the average model fitness score rises less than 1 % and drops less than 0.5 % over the last 5 iterations).

As mentioned before, the output of the algorithm is a ranked set of models, called the result population. Initially, the result population is filled with the models from the random population, removing those that are contained by one or more other models in the result population. A model is contained by another model when the other model allows everything that the first model allows and more. For example, the left model is contained by the right in Fig. 2. New valid models generated in iterations are added to the result population continuously. Every time new models are added, the result population is sorted based on the fitness function. If the population gets bigger than its specified maximal size, than the bottom models are removed to make it fit again.

Note that the proposed population-based meta-heuristic does not use recombination. The target environment is very complex, so finding new valid models starting from the valid models in the random population is always going to be hard. However, the chances of finding such a model are arguably higher when using a sequence of mutations than by randomly combining two models. This is because mutations entail relatively small changes to the model, whereas recombination is more disruptive. It is thus a choice between incremental versus disruptive change. The disruptive character of recombination offers as advantage a high diversity of models found. This is why it is important to add random seeds from the random population to the edit population

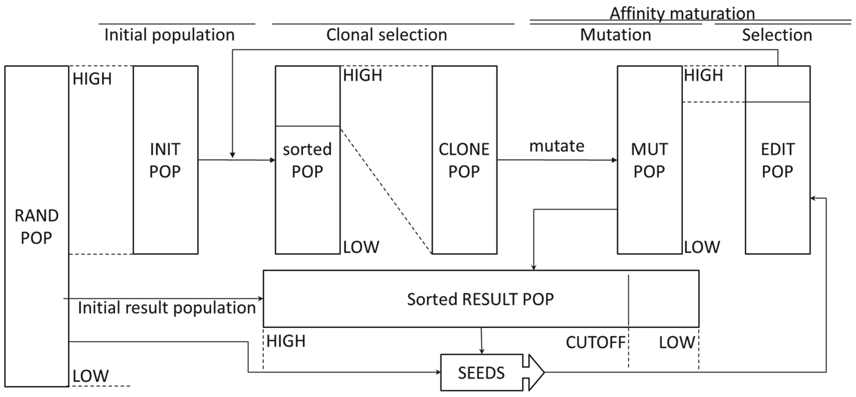


Fig. 1. Proposed artificial immune system implementation (based on Fig. 1 from [7])

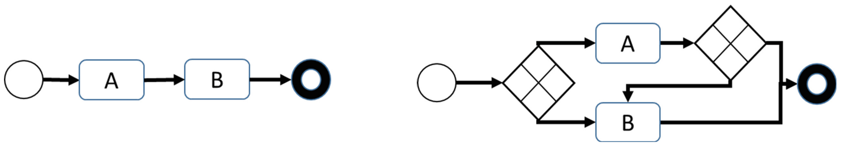


Fig. 2. Illustration of contained (left) and containing (right) imperative models

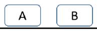
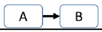
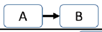
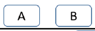
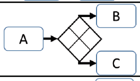
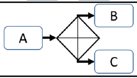
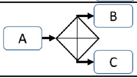
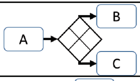
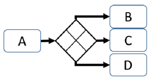
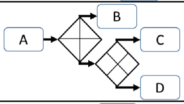
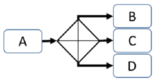
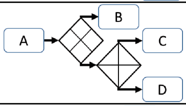
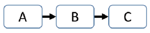
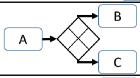
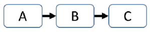
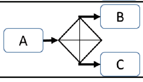
during iterations in the AIS technique. This ensures that new models are being searched in all search directions, which means that diversity is also ensured provided that the random population is in fact generated at random (special attention was paid to this in the implementation of the proof of concept).

Clonal selection. The B-cells of the vertebrate immune system use receptor molecules (i.e., antibodies), present on their surface, to bind with molecules covering the antigens (i.e., epitopes). They are cloned proportional to the degree to which the B-cell can recognize a certain antigen. The composition of the clone population (Fig. 1) tries to mimic this. The best p_{clone} percentage (e.g., 60 %) of models in the input population are cloned and put in the clone population. To make this population the same size as the input population, multiple clones of the same model are allowed. The fitness function of a model determines the chance that that model is chosen to be duplicated in the clone population (more than twice is possible).

Mutation (hypermutation). Random changes are made to the variable region of the receptor molecules of B-cells. The higher the degree to which the B-cell can recognize the target antigen, the lower the mutation rate and vice versa. Similarly, the models in the clone population are mutated (see Table 1) and then added to the mutation population (Fig. 1). The amount of mutations per model is determined by its fitness score. The models will be mutated at least once, possibly more. The chance that a model is mutated more is inversely proportional to its fitness score.

Note that models in the mutation population are not necessarily valid. This is not a problem because invalid models could become valid again after more mutations in

Table 1. The proposed set of mutations

	Original	Mutation
Activate empty transition		
Remove active transition		
Exclusive to parallel		
Parallel to exclusive		
Add exclusive to parallel		
Add parallel to exclusive		
Serial to exclusive		
Serial to parallel		

subsequent iterations or are filtered out since they are assigned the minimal fitness score of zero. All valid mutation found are also added to the result population. Like before, all doubles and contained models are removed from the result population. It is then sorted and the worst models are removed until the result population has again its intended size.

Selection (receptor editing). The mutations to the cloned B-cells cause many to become useless due to a bad combination of mutations. These non-functional B-cells are removed from the population by a programmed cell death (i.e., apoptosis). The last intermediate population, called the edit population (Fig. 1), does the same with the models in the mutation population. It is partially filled with the best p_{edit} percentage (e.g., 50 %) of models from the mutation population (again, invalid models are possible). The remainder of the edit population is filled with randomly selected seed models. Half of these seed models are taken from the random model, to keep searching in all directions, and the other half from the result population, to possibly optimize already found results.

Fitness function. The fitness function is used to estimate the value of an imperative process model multiple times during each iteration of the algorithm. The absolute fitness score of a model is not important; it is only of relative importance compared to scores of the other models in the population during one execution. It would thus be incorrect to directly compare the fitness scores of models from different executions of the algorithm (even if they start from exactly the same declarative process model). In this subsection we will discuss the three weighted components that are used: overall completion time, build-time flexibility and run-time flexibility. The actual weights used in the proof of concept (see Sect. 3) are based on our perceived importance of each subscore, respectively 20 % - 40 % - 40 %, but are still open for discussion based on the specific application environment.

- Overall completion time: the optimization goal we have pursued in our current implementation (but others can be used instead or in combination). It represents the time needed to complete the whole process using this model. If each activity is executed exactly once, then the overall completion time can easily be determined exactly. But when other existence-templates of the Declare model are used, we can only estimate how many times a certain activity will be executed. For example, 'Existence3(A)' specifies that activity A is executed at least 3 times with no upper bound. Combining this with the given estimates of the duration of an activity (e.g., 'Duration(A) = 5'), an estimate of the completion time of the model can be calculated. Since it is not always known exactly how many times an activity will be executed, the variable bound (e.g., the 3 in Existence3) of the template will be used as the number of times used in the estimation of the overall completion time. Machine learning algorithms will be used to offer a better estimation in the final system.
- Build-time flexibility: the inherent degree of flexibility of the model itself at build-time, also known as the looseness of the model [2]. Processes with a high degree of looseness are processes where the goal is known a priori, but a high degree of freedom is given on how to achieve it. The respective score is determined by counting the number of transitions from one activity to another allowed by the model, divided by the total number of transitions allowed by the input declarative model.
- As an example the left and right model from Fig. 2 will be scored in the context of a declarative model allowing six transitions (start-A, start-B, A-B, A-end, B-A and B-end). The left model has three transitions: start-A, A-B and B-end. This means that the build-time flexibility score is 3/6. The right model on the other hand has a build-time flexibility score of 5/6: start-A, start-B, A-B, A-end and B-end. This reflects that in the left model allows only one path (e.g., start-A-B-end), whereas in the right model three paths (e.g., also start-A-end and start-B-end) can be followed.
- Run-time flexibility: a property of a process instance according to [2]. In our case, the process instance is actually a recommendation system based on a declarative model. This is what is deployed, and thus, this is the context in which it has to be measured. The flexibility offered by the system depends on the result population, because only next steps contained by this population are sorted. The remaining steps are still selectable (requiring recalculation to check feasibility and generate a new result population), but not recommended. This means that run-time flexibility is not scored on an individual model level like the previous two scores, but rather relative to the models in the result population. Our proposition is to score the run-time flexibility with two conflicting scores, so that a balance has to be reached.

When scoring a model compared to a model from the population, the first score represents the number of activities at which one could switch to the other model, divided by the total number of activities (defined in the input declarative model). It is possible that for a certain activity some cases allow a switch and other cases don't. If 0 represents a no scenario where no switch is possible and 1 represents that a switch is always possible, then partial points represent the number of cases in which it is possible, relative to the all cases. The total aggregated score of a certain model compared to the population allows us to valuate if the model can be used when diverging from another model to this model, or when diverging from this model, there are other models in the set that can be used.

It is important for the correct calculation of the first score that models that are contained by other models are removed from the result population before evaluating them based on this component. If not, it causes the component to estimate the value of the models incorrectly. The scores of the containing and contained models will be higher than they should be, because of the similarity between them, but have no real value since switching between these models is pointless.

A limitation to the first score is that it has a preference for similar models with different endings, since this allows the maximal amount of switching between them. This means that certain variations at the beginning of the model will only have a small chance to be in the population. To counteract this tendency, the second part of the run-time flexibility score is inversely proportional to the frequency of transitions used in a model compared to the population. This tends to equalize the diversity and thus balances out the total run-time flexibility score.

3 Demonstration

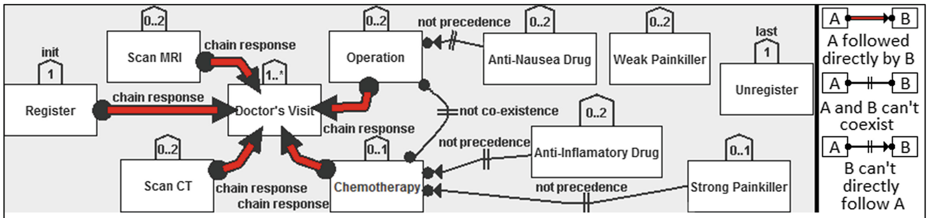
In this section, a brief demonstration of the recommendation system will be given. The system itself has not been fully implemented yet, but the proposed algorithm[1] has. The demonstration will use the simplified cancer treatment center described in the introduction. Patients initiate the process by registering at the reception. The next activity is a doctor's visit to evaluate what has to happen next (e.g., order scans, operate…). The only other certainty is that the patient will have to unregister at the end. The example environment is represented by the Declare model, with the given estimates for the durations, described in Fig. 3. This model is complex enough to highlight the basic aspects of the proposed recommendation system. This model is used as input for the algorithm proposed in this paper. The result is a set of imperative models, 30 models in this case, that comply with the given declarative process model.

Based on the sorted set of models in the result population, a sorted set of recommendations is created (see Table 2). This possibly leaves the user with one or more possible next steps that are not present in any model in the result set (i.e., steps that severely limit the flexibility later on). These are placed below the lowest recommendation, as they are not really recommended, but could still be chosen in rare cases. Impossible next steps are of course disregarded. Parallel next steps are currently not supported by the recommendation system (but could possibly be in the result population), so they will not be included in this demonstration. The first column of Table 2 contains the activity trace that is followed during this demonstration. A row contains

[1] https://github.ugent.be/MIS/AIS_Population_RecommendationSystem/.

the enumeration of the possible next activities. The content of each box indicates if a certain next activity is a recommendation or not at that time. Initially, only the first row is given. The choice made then adds the chosen activity to the trace and a new row is added with recommendations for the next step. The numbers represent the ranking of the recommendations. A lower rank is preferred, while equal rank means that both next steps are allowed by the same model. Unranked recommendations are marked with a dash (-) and impossible next steps are marked with an X.

The process starts with two activities with no alternatives (and thus no other recommendations). The patient registers at the reception of the cancer treatment center and then he is examined by the doctor. The doctor decides that a CT-scan is needed. He thus follows one of the top recommendations. Right after the session, another examination is performed by the doctor to check if everything is in order. However, the



Duration(Register)=4, Duration(Unregister)=2, Duration(Scan MRI)=23, Duration(Scan CT)=17, Duration(Operation)=90, Duration(Chemotherapy)=19, Duration(Weak Painkiller)=1, Duration(Strong Painkiller)=1, Duration(Anti-Nausea Drug)=1, Duration(Anti-Inflammatory Drug)=1, Duration(Doctor's Visit)=10

Fig. 3. Example Declare model with given estimates of duration of each activity

Table 2. Example of an execution trace of the recommendation system.

	Register	Unregister	Scan MRI	Scan CT	Chemotherapy	Operation	Weak Painkiller	Strong Painkiller	Anti-Nausea Drug	Anti-Inflammatory Drug	Doctor's Visit
	[✓ 1]	X	X	X	X	X	X	X	X	X	X
Register	X	X	X	X	X	X	X	X	X	X	[✓ 1]
Doctor's Visit	X	1	1	[✓ 1]	1	18	1	-	1	-	1
Chemotherapy	X	X	X	X	X	X	X	X	X	X	[✓ 1]
Doctor's Visit	X	1	1	1	1	[✓ 18]	1	-	1	-	1
Operation	X	X	X	X	X	X	X	X	X	X	[✓ 1]
Doctor's Visit	X	1	1	1	1	5	1	[✓ -]	-	-	1
	RECALCULATION										
Strong Painkiller	X	[✓ 1]	1	1	X	2	1	X	1	1	1
Unregister											

doctor notices that something is wrong and concludes that the patient needs instant surgery. Even though this is not the top recommendation, another model in the population (ranked 18th of the 30 original models) does allow the next activity to be an operation. The operation is successful and the doctor gives him a strong painkiller to ease the pain. At this point, there is no model left in the population (as models are removed that do not allow the current chosen activity trace) that allows the next activity to be the prescription of a strong painkiller. This means that the algorithm has to recalculate. After some time the patient feels ready to go home, so he goes back to the reception and unregisters himself from the cancer treatment center. The newly calculated result population has this last step now as one of the top recommendations. This step is also the end of the process.

4 Optimization Fitness Function

As was described before, the fitness function is an estimation of the value of a certain model that can be used to compare models during one execution of the proposed algorithm. It was also mentioned that models from the population contained by other models from the population are removed, because they negatively impact the valuation made by the fitness score. This, however, applies not only to containing models in the population, but also containing models that have not been discovered yet. Theoretically, this has no effect on the outcome of the algorithm since contained models will eventually be removed when the containing model is found. Nonetheless, in practice processing time is limited and thus it is possible that they are never discovered. This causes a systematic overestimation of the fitness score (as similarities between contained models are high). Also, when a containing model is found, one or more models are suddenly removed from the result population. This causes the average fitness score to possibly rise or drop significantly based on the composition of the result population as can be seen in Fig. 4 (between iteration 145 and 160 the population drops from 30 to 13, resulting in a big drop in the average fitness score). The other side effect is that previously disregarded models now suddenly could have become relevant. These models where however forgotten by the algorithm, so they have to be rediscovered, slowing down the convergence (as it takes about 40 iterations to get back to 30 models in the result population in Fig. 4).

An optimization step is proposed to handle the issue, which is executed before the fitness scores are calculated. In this step an attempt is made to combine each pair of models in the result population.[2] If there exists a valid model containing both models, it has to contain at least the union of both these models, since it is know that each model in the result population is valid. So if the union is valid, a containing model of both models has been found. This filters out all models contained by other models in the population (as there union will be equal to the containing model from the population), but it also discovers new models that contain one or more models in the population.

[2] This is not the same as recombination, as the result is not a composition of smaller parts of each model. The result is a model that contains both models fully.

By repeating this step each time models are added to the result population, no significant errors will be made during the valuation of the models. It also eliminates the drops in size of the population, giving the average fitness score of the result population a smoother progression during iterations (see Fig. 5).

Note that it is still possible that a certain model in the population is contained by another valid model not in the population. This is not a problem because the fitness scores will not be negatively influenced by this, since it concerns just one model in the population. The containing model can be found with a certain combination of mutations. At which point the containing model will just replace the contained model in the result population. So similarities between multiple containing/contained models do not come into play.

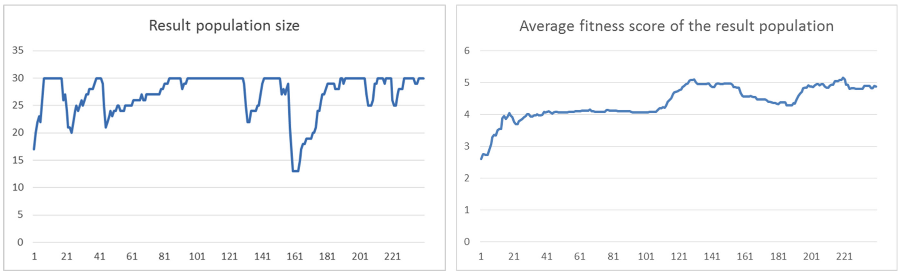


Fig. 4. The effects of the systematic removal of intrapopulation contained models on size and average fitness score (X-axis) of the result population during iterations (Y-axis).

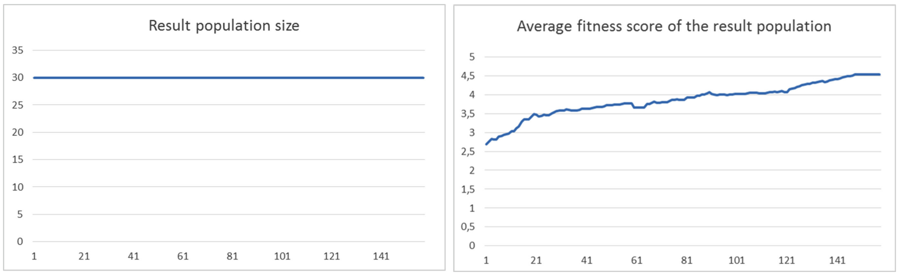


Fig. 5. The size and average fitness score (X-axis) of the result population with the optimized algorithm during iterations (Y-axis)

5 Conclusion

This paper presents a recommendation system based on an AIS algorithm to generate imperative process models from a declarative model. In contrast to existing recommendations systems, this one is made to thrive in dynamic and complex environments with high flexibility needs. This requires a focus on the flexibility of a process instead of just the optimization goals. This flexibility is divided along two dimensions:

build-time and run-time. Build-time flexibility is an intrinsic property of a model, which is measured by comparing the number of transitions allowed by the imperative model to the number of possible transitions allowed by the declarative model. Run-time flexibility is the flexibility allowed after deployment. A new measure of the run-time flexibility was introduced, suited for this context and made possible by the proposed algorithm. The proposed algorithm also has several other advantages:

– Instead of returning one optimal model to be used as recommendation, the proposed algorithm returns a ranked set of models. This reduces the need to recalculate when the user diverges from the top recommendation.
– The original result population will contain less valid models as execution progresses because some models will become invalid by the choices made. A threshold can be used to determine if the population has become too small relative to size of the road ahead. Recalculations can be faster than normal calculations when using the remaining valid (or even the invalid) models as start population (possibly supplemented with some newly generated models).
– Small changes to the environment (e.g., adjusting estimates for the duration of activities) can often be handled by re-sorting the set of models. If there are still enough valid models (based on a threshold mentioned above) in the set no recalculations are needed. When recalculations are needed, these will be faster depending on how many valid models remained from the previous population.

This paper is only the starting point of this research. The idea for the robust and recommendation-based process engine has been fully developed and its implementation has begun. An implementation of the algorithm has been made that proves its usefulness, whereas the implementation of the recommendation system in a whole is still a work in progress. But first off, a thorough evaluation of the implemented algorithm is needed. This should reveal the optimal parameter values for the algorithm (i.e., fitness function weights, population sizes…) generally or in specific cases. These parameter values are also needed to allow a correct evaluation of the performance of the algorithm. This can be combined with a theoretical comparison of the technique to other techniques to determine its performance and usability. Additionally, an evaluation in real life cases can provide clear guidelines for which situations tend to be most suitable for using this technique. Finally, further research can also be done in how to improve the technique by adding resource constraints and combining it with decision models, machine learning techniques and/or process mining techniques.

References

1. Weske, M.: Business Process Management: Concepts, Methods, Technology. Springer, Berlin (2007)
2. Reichert, M., Weber, B.: Enabling Flexibility in Process-Aware Information Systems. Springer, Heidelberg (2012)
3. Van der Aalst, W.M.P., Pesic, M., Schonenberg, H.: Declarative workflows: Balancing between flexibility and support. Comput. Sci. Dev. **23**, 99–113 (2009)

4. Schonenberg, H., Weber, B., van Dongen, B.F., van der Aalst, W.M.: Supporting flexible processes through recommendations based on history. In: Dumas, M., Reichert, M., Shan, M.-C. (eds.) BPM 2008. LNCS, vol. 5240, pp. 51–66. Springer, Heidelberg (2008)
5. Barba, I., Weber, B., Del Valle, C., Jiménez-Ramírez, A.: User recommendations for the optimized execution of business processes. Data Knowl. Eng. **86**, 61–84 (2013)
6. Goedertier, S., Vanthienen, J., Caron, F.: Declarative business process modelling: principles and modelling languages. Enterp. Inf. Syst. 1–25 (2013)
7. Van Peteghem, V., Vanhoucke, M.: An artificial immune system for the multi-mode resource-constrained project scheduling problem. In: Cotta, C., Cowling, P. (eds.) EvoCOP 2009. LNCS, vol. 5482, pp. 85–96. Springer, Heidelberg (2009)