Using Monotonicity to find Optimal Process Configurations Faster

D.M.M. Schunselaar[1⋆], H.M.W. Verbeek[1⋆], H.A. Reijers[1,2⋆], and W.M.P. van der Aalst[1⋆]

[1] Eindhoven University of Technology,
P.O. Box 513, 5600 MB, Eindhoven, The Netherlands
{d.m.m.schunselaar, h.m.w.verbeek, w.m.p.v.d.aalst, h.a.reijers}@tue.nl
[2] VU University Amsterdam,
De Boelelaan 1105, 1081 HV Amsterdam, The Netherlands
h.a.reijers@vu.nl

Abstract. Configurable process models can be used to encode a multitude of (different) process models. After configuration, they can be used to support the execution of a particular process. A configurable process model represents a space of instantiations (configured process variants). Such an instantiation space can be used by an organisation to select the best instantiation(s) according to some Key Performance Indicators (KPIs), e.g., cost, throughput time, etc. Computing KPIs for all the instantiations in the space is time consuming, as it might require the analysis (e.g., simulation) of thousands (or more) of instantiations. Therefore, we would like to exploit structural characteristics to reduce the amount of instantiations which need to be analysed. This reduction only removes those instantiations which do not need to be considered by an organisation. This yields the same result (a collection of best configurations), but in a faster way.

Keywords: Configurable Process Model, Business Process Performance, Analysis, Monotonicity, Petra

1 Introduction

To go abroad, travellers typically need to have a passport. To obtain a new passport, a traveller needs to go to his own municipality. For example, Fry has to go to the New New York municipality, while Homer has to go to the Springfield municipality. Although the New New York municipality will create a passport for Fry, they will not do so for Homer, as he is not living in New New York, but in Springfield. As such, the municipalities all offer the service to create a

⋆ This research has been carried out as part of the Configurable Services for Local Governments (CoSeLoG) project (http://www.win.tue.nl/coselog/).

new passport, but they are not competing with each other as they only offer this service for their own inhabitants. As a result of the latter, municipalities are quite eager to share their processes with other municipalities, and to learn from each other. Although all municipalities offer the service to create a new passport, they do not all use the exact same process. Some "couleur locale" may exist between different municipalities. For example, the New New York municipality may create the passport first and have Fry pay when he collects his passport, while the Springfield municipality, being smaller, may require Homer to pay when he requests for it, that is, before they create it. Even though the steps in the process may be the same (request for a passport, pay for it, create it, and collect it), the process may still differ to some extent.

The combination of this "couleur locale" and the openness mentioned earlier makes municipalities natural candidates to benefit from co-called configurable process models. A configurable process model contains variation points which can be set to tailor the configurable process model to the preferences of an organisation. Setting preferences for the variation points is called a configuration. If all variation points are set by a configuration, the resulting process model is called an instantiation. Please note that a configurable process model that contains a multitude of variation points allows for an exponential amount of possible configurations and instantiations.

A configurable process model may contain instantiations that are not used by any of the municipalities. An interesting question now is, whether some instantiations score better for a given municipality than the instantiation they are now using. Given a set of Key Performance Indicators (KPIs), we could check how every instantiation scores on these KPIs, and return the corresponding best configurations. In [1], we have introduced *Petra*, a generic framework that automatically analyses KPIs on all instantiations of a configurable process model for instance by simulating all the instantiations. By exhaustively searching *all* instantiations, *Petra* returns a Pareto front [2] of the best instantiations to the municipality, which can then select the configuration they like best.

As mentioned earlier, a configurable process model may allow for many variation points and, as a result, very many configurations and instantiations. As a result, the amount of instantiations may be too large to analyse, or it may simply take too much time to analyse them all. Therefore, in this paper, we aim to reduce the amount of instantiations which need to be analysed by exploiting the fact that they all stem from the same configurable process model. For example, if we take the passport example as introduced earlier, it is clear that the Springfield instantiation allows for less financial risks than the New New York one. For this reason, if financial risk would be the KPI at hand, it would make no sense to analyse the New New York instantiation, as it will be dominated anyways by the Springfield instantiation.

To achieve this reduction in the amount of instantiations to analyse, we propose to exploit structural properties of the configurable process model by means of a *monotonicity* notion. This paper introduces this monotonicity framework, applies it for a concrete KPI, and evaluates the application empirically on an

artificial configurable process model. The monotonicity framework creates, per KPI, an ordering of the instantiations. This ordering starts with the instantiation most probably to have a high score on that KPI. Using these orderings, the monotonicity framework starts analysing the most promising instantiations. It keeps on analysing until no more instantiations can be found which score higher on a KPI than the already analysed models.

For our concrete KPI, we have selected throughput time and we show how the monotonicity can be computed between two instantiations. We apply this monotonicity notion on a running example to order the instantiations. Afterwards, we use simulation to obtain concrete values for the KPI. Using these values, we can conclude that we achieve a reduction of more than 90% on the amount of instantiations which need to be analysed for this particular model.

The remainder of this paper is organised as follows. In Sect. 2, we elaborate on related work. Afterwards, we present some preliminaries in Sect. 3. Sections 4 and 5 contain our monotonicity framework and concrete results for the concrete throughput time KPI. We finish the paper with an empirical evaluation and the conclusions and future work in Sect. 6 and Sect. 7.

2 Related Work

Our research builds on three existing lines of research: configurable process models, performance analysis, and business process reengineering.

2.1 Configurable Process Models

Configurable process models (see Sect. 3.1 for an example configurable process model) have been developed by extending existing modelling languages, e.g., C-EPC (Configurable Event-driven Process Chain) [3], C-BPEL (Configurable Business Process Execution Language), and C-YAWL (Configurable Yet Another Workflow Language) [4]. A more complete overview of variability support is provided in [5]. All these approaches are mainly focussed on supporting variability and not so much on the analysis of the resulting instantiations.

2.2 Performance Analysis

Within the field of queueing theory, work has been conducted in defining monotonicity notions between queueing networks and between queueing stations. In [6], the author defines a notion between queueing stations and between queueing networks for closed queueing networks. The definition of monotonicity employed is similar to our notion of monotonicity. However, this paper is mainly focussed on the parameters of the network (number of jobs, processing speed of networks) and not on the relation between two topologically different networks.

The authors in [7] consider performance monotonicity on continuous Petri nets. Similar to the work in [6], the authors consider monotonicity in terms of the parameters of the Petri net and not in terms of the structure of the Petri

net. However, since our used formalism can be translated to Petri nets, this is an interesting approach to consider for future work.

Although the papers consider monotonicity in a similar way as us, they focus on the parameters instead of the topology of the network. However, one of our future directions is to take the parameters also into account in which this work might be applicable.

2.3 Business Process Reengineering

In [8], the author presents a tool KOPeR (Knowledgebased Organizational Process Redesign) for identifying redesign possibilities. These redesign possibilities are simulated to obtain performance characteristics such that they can be compared. The analysis of process models focusses mainly on the analysis of a single model and not on various instantiations. An approach evaluating when certain changes to the structure of the process model are appropriate is presented in [9]. The paper starts from a number of commonalities in reengineered business processes and deduces, based on queueing theory models, under which circumstances a change to the structure of the process model is beneficial. Some ideas of their paper can be applied to our setting but the majority of ideas is not tailored towards throughput time.

In [10], so-called Knock-Out systems are discussed and heuristics are defined for optimising these. Similar to [9], heuristics are defined and formally shown if it is beneficial to apply a certain heuristic in a particular setting. Their approach allows for more flexibility in defining the processes than configurable process models, i.e., in the paper only a precedence relation is defined between tasks. At the same time, the processes considered are less flexible as they do not include choices, i.e., every task has to be executed. As with the previous approach, some ideas can be used in our approach.

The approach closest to our approach is presented in [11]. In their approach, various process alternatives are analysed. These alternatives are obtained by applying redesign principles instead of starting from a configurable process model. Their approach can benefit of the work presented here as it might remove the need to analyse some instantiations due to monotonicity.

3 Preliminaries

Before introducing the monotonicity framework, this section introduces the configurable process models used by the framework and the *Petra* framework.

3.1 Configurable Process Models

Configurable process models contain predefined variation points. By configuring these variation points, that is, by selecting appropriate values for these points, the configurable process model can be tailored to an organisation (like a municipality). If all variation points have been configured properly, that is, if the

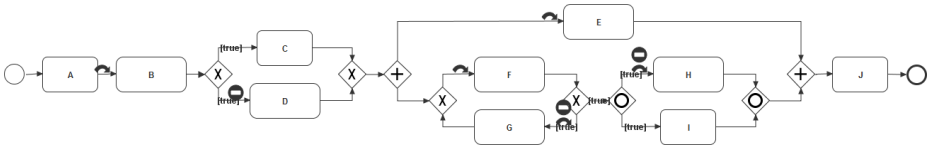

Fig. 1: Running example of a configurable process model in BPMN notation.

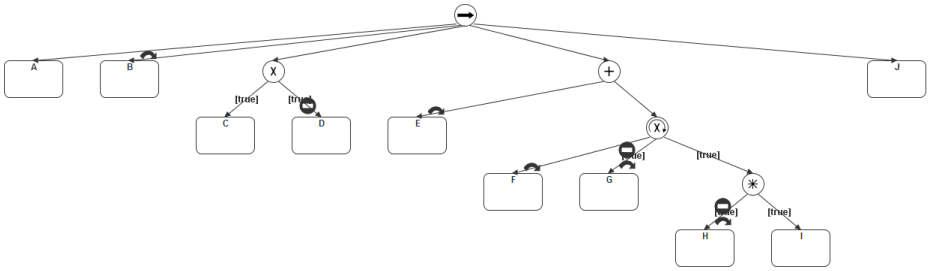

Fig. 2: Running example from Fig. 1 in Process Tree notation.

configuration is complete, the configurable process model is instantiated into a process model ready that is ready for enactment (an instantiation).

As an example, Fig. 1 shows the control-flow of a configurable process model using a BPMN (Business Process Model and Notation) notation augmented with curved arrows and no-entry signs. A curved arrow on an incoming edge indicates a variation point that allows the following part of the configurable process model to be hidden. For example, the curved arrow on the incoming edge of the task labelled "B" indicates that this activity can be hidden. Note that if this curved arrow would have been positioned on the outgoing edge of this task, that then the entire following choice part, including the tasks labelled "C" and "D", would then have the option be hidden. Likewise, the no-entry sign indicates a variation point that allows the following part of the configurable process model to be blocked.

If a part of the configurable process model is hidden, this results in this part being substituted by an automatic task. If a part of the configurable process model is blocked, this results in removing this part in total. As a result, if a part of a sequential execution of tasks is blocked, then the entire sequence is blocked, as we would run into a deadlock otherwise.

For our configurable process model formalism, we use so-called Process Trees [1]. Figure 2 shows the Process Tree representation of the process model as shown in Fig. 1. A Process Tree is a block-structured process modelling formalism and is specifically developed in the CoSeLoG project. The main advantage of Process Trees over other formalisms is that it ensures soundness, e.g., there cannot be any deadlocks [12]. The various node types in the Process Tree and their semantics in BPMN are depicted in Fig. 3. Nodes come in three flavours; tasks, blocks, and events. Tasks form the units of work and can be either automatic (without

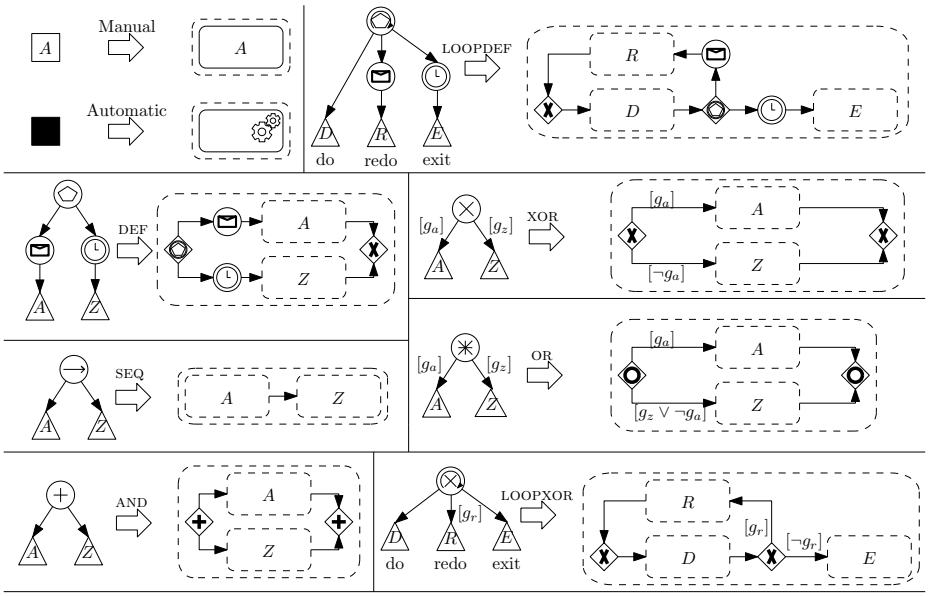

Fig. 3: The various nodes of a Process Tree and their BPMN semantics. For choices we have a default option being the last child. This means that if all expressions evaluate to false the last branch is chosen. For the XOR, this is encoded by annotating the last branch with the negation of the other branch. Note that we have shown the case with two children. It is trivial to extend this to more children.

a resource) or manual (with a resource). Blocks indicate the causal dependency of the children, i.e., the nodes directly underneath the block. Events indicate a point in the process where input from the environment is required. Only events that match the actual input will be executed, the other events will be dropped. In principle a block can have any number of children except for the LOOP nodes (LOOPXOR and LOOPDEF), which always have 3 children, and the EVENT nodes, which have a single child. At the top of the Process Tree we have a root node.

Next to hiding and blocking, a Process Tree allows for a third type of variation points, called placeholder nodes. Where hiding and blocking can be configured by selecting either "yes" or "no", a placeholder node can be configured by selecting one of its child nodes. As a result, the placeholder node will be replaced in an instantiation by its selected child node. For instance, in a configurable process model, there may be the possibility to select a payment method from a set of known payment methods (credit card, bank transfer, or cash). The Process Tree formalism is richer than just the control-flow perspective [1], but here we limit ourselves to the control-flow perspective, i.e., we assume the other perspectives remain unchanged.

3.2 Petra

Petra [13] (Process model based Extensible Toolset for Redesign and Analysis) is a framework for analysing configurable process models. *Petra* employs an iterative brute force approach in traversing the instantiations of a configurable process model. In each iteration, *Petra* chooses an instantiation and applies various analysis tools to this. As a result, the values for the required KPIs become known for this instantiation, and this instantiation can be added at the specific point in the Pareto front [2]. As the Pareto front only keeps track of the best (non-dominated) points, instantiations that may have been added may be removed at some point in time. In the end, only those instantiations that are not dominated by some other instantiation will survive on the Pareto front. The sets of tools and KPIs within *Petra* are extensible with new tools and KPIs. As a result, we prefer not to limit the monotonicity notion to a predetermined set of KPIs.

4 Monotonicity Framework

The monotonicity framework stems from the observation that it may be possible to check whether an instantiation dominates another instantiation by comparing the structure, control flows in or case, instead of the behaviour. This means that we want to compare two models with respect to a KPI without computing the actual values for that KPI. Consider, for example, two instantiations from the running example (see Fig 2), and assume that for the first instantiation nothing has been blocked or hidden, and that for the second nothing has been blocked and only the task labelled "B" has been hidden. Clearly, the throughput time of the second instantiation is always better than the throughput time of the first, as the only difference is that the first has to execute "B", which we assume takes some time, where the second does not. Based on structure, we could claim that the second instantiation is better as the first w.r.t. the throughput KPI under all circumstances. As a result, when looking only at the throughput KPI, the first cannot be better than the latter (assuming independence between the duration of an activity and the occurrence of another).

For this reason, the monotonicity framework introduces an acyclic (partial) order on the possible instantiations. Throughout the paper, we use the term "at-least-as-good" to denote the monotonicity ordering between instantiations/nodes for a particular KPI. We define this relation formally as:

Definition 1 (At-least-as-good). *A node n is at-least-as-good as another node n' (denoted $n \geq n'$) w.r.t. KPI K if $\forall_c P[K(n) \leq c] \leq P[K(n') \leq c]$, i.e., the cumulative distribution function (CDF) of the distribution $K(n)$ is for every point c* at most *the CDF of the distribution of $K(n')$ in that point. An instantiation M is at-least-as-good as another instantiation M' w.r.t. KPI K if and only if the root node of M is at-least-as-good as the root node of M' w.r.t. K.*

Note that it is possible that individual values of $K(n')$ are better than $K(n)$, but overall the values for n are better than n'.

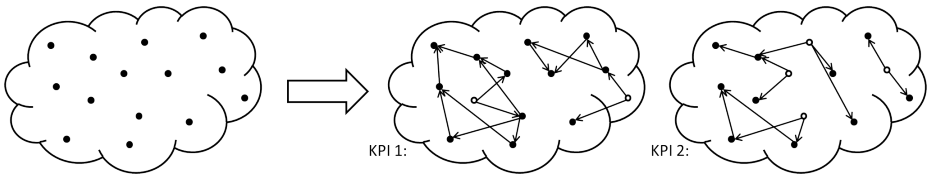

Fig. 4: Using monotonicity, we can transform the set of possible instantiations to a partial order indicating the most promising order of analysing the instantiations for various KPIs.

If we have that a model M is at-least-as-good as a model M' for all (relevant) KPIs, then clearly M should dominate (or at least equal) M' and as a result it only makes sense to analyse M' if M needs to be analysed. Later on, we will see how we can derive this at-least-as-good relation. First, however, we will show how *Petra* uses this relation.

Graphically, the monotonicity transforms the collection of possible instantiations to a collection of possible related instantiations. Since there might be multiple KPIs in the framework, we obtain (different) relations for each of the KPIs (Fig. 4). The dots are the instantiations and an arrow between two dots indicates that the instantiation at the tail of the arrow is at-least-as-good as the instantiation at its head. The open dots indicate the most promising instantiations. By transitivity, if there exists a directed path from one instantiation to another instantiation, then the former instantiation is at-least-as-good as the latter w.r.t. the corresponding KPI.

With the monotonicity framework added, *Petra* analyses the possible instantiations in a specific order. If M is at-least-as-good as M' on all KPIs, then M will be analysed by *Petra* before M' will be analysed. If M' is to be analysed by *Petra* and if at that point in time M has been dominated by some other model M'', then M' cannot dominate M'' and there is no use in analysing it. Otherwise, if M is not dominated, M' is analysed by *Petra*.

When an instantiation is dominated by other models, it creates a cut-off point along the partial orders for the various KPIs, as this instantiation is at-least-as-good (w.r.t. the KPI at hand) as every instantiation that can be reached by a directed path. As a result, if an instantiation is below the cut-off points for all KPIs, it is dominated by the previously analysed models and there is no use in analysing it.

To determine whether an instantiation is at-least-as-good as another instantiation, we need to check whether its root node is at-least-as-good-as the other root node. To determine this, we use a bottom-up approach, which uses the fact that both are instantiations of the same configurable process model. As a result of this, we can relate two nodes in both instantiations in a straightforward way by determining whether they stem from the same node in the configurable process model: They are related if and only if they stem from the same node. Note that hiding a part of the configurable process model results in an automatic task

with duration 0 in the instantiation. As a result, such an automatic task can be related to any other node. Furthermore, note that if a placeholder node is configured as a node of one type (like SEQ) in one instantiation, and as a node of another type (like AND) in another instantiation, it is possible that a node of one type is related to another node of another type.

For a task node, it is usually quite straightforward to check whether or not they are at-least-as-good as their related nodes: As both stem from the same node, they are equal, and hence at-least-as-good. For a block node, we need to look whether all relevant child nodes are at-least-as-good as their related nodes, which is where the bottom-up approach comes in. Based on the structures of both instantiations and using the fact which child nodes are at-least-as-good as their related nodes, we determine whether a give node in one instantiation is at-least-as-good as its related node in the other instantiation. With this approach, we decompose the problem into smaller problems and basically use patterns to identify which of the elements is better. If we can conclude that the one root node is at-least-as-good as the second root node, then we can conclude that the one process model is at-least-as-good as the second process model.

In our monotonicity framework, we assume there is a correspondence between node types and the effects on the value of a KPI. This stems from the observation that some KPIs behave monotone in two flavours, i.e., more is better (monitoring activities in the process for compliance) or less is better (costs). KPIs which do not behave monotone in that respect, e.g., wait time which can increase and decrease with the addition/removal of activities, cannot be captured in this framework.

Because we only take the structure of the instantiations into account, we restrict ourselves to the control-flow perspective in this paper. However, to be able to compare choice nodes (XOR, DEF, OR), we assume there is a probability associated with the outgoing edges of the choice node. This probability indicates the (relative) probability that the flow of control follows that path. Likewise, to be able to compare LOOP nodes (LOOPXOR and LOOPDEF), we assume that there is a probability associated with the outgoing edges to the REDO and EXIT blocks. In the next section, we demonstrate the applicability of our approach by focussing on a single KPI.

5 Throughput Time

The example KPI to be used in our monotonicity framework is the throughput time (sometimes also called sojourn time or lead time). The throughput time is the time it takes a case from start to end. We have chosen the throughput time since this KPI is well-studied and often considered for analysing process models. Using our monotonicity framework, we need to be able to take two instantiations (two Process Trees) and decide which instantiation is at-least-as-good (if any). As mentioned, we focus on the control-flow perspective. Therefore, we assume the other perspectives do not change between different instantiations. However, we do not disregard the other perspectives as this might lead to counter-intuitive

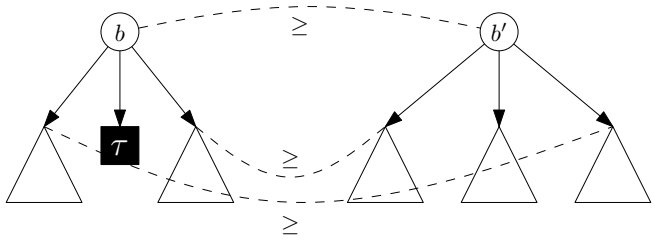

Fig. 5: The general constraints for comparing two block nodes. Note that if the child of b is a silent task (black square with white τ), it does not need to be related to a child of b'.

results, e.g., if we have a choice between a fast and a slow branch, then reducing the amount of work for the slow branch and increasing the amount of work for the fast branch might actually increase the throughput time since the fast branch cannot handle more work. Therefore, we focus on reducing the amount of work for branches without increasing the amount of work for other branches. We go through the collection of nodes and present when a node is at-least-as-good as the related node (w.r.t. throughput time).

5.1 Tasks

A silent task (an automatic task with duration 0) is always at-least-as-good as any node, and can be ignored (when not related) in a SEQ or AND block. Any other automatic task can be compared according Def. 1 with another automatic task. In all other cases, we cannot say whether an automatic task is at-least-as-good as the other node.

A manual task is at-least-as-good as the same manual task. In all other cases, we cannot say whether a manual task is at-least-as-good as the other node. Please note that, as mentioned earlier, we only take the control-flow perspective into account. If we would take the resource perspective into account, then we could check whether the same manual task would be performed by generally faster or less overloaded employees.

5.2 Blocks

In Fig. 5, the general case is depicted where every comparison between block nodes has to adhere to. A block node b is at-least-as-good as a related block node b' if every child node c of b (except for silent tasks) is related to a child node c' of b' such that node c is at-least-as-good as node c'.

The general case is sufficient for the SEQ, AND, and EVENT nodes which are related to nodes of the same type. Next to this, if b is an AND node and b' is a SEQ node and the general case holds, then we can also conclude that b is at-least-as-good as b' since doing things is parallel is at-least-as-good as doing things in sequence for the throughput time. Finally, if b is a SEQ or AND node and

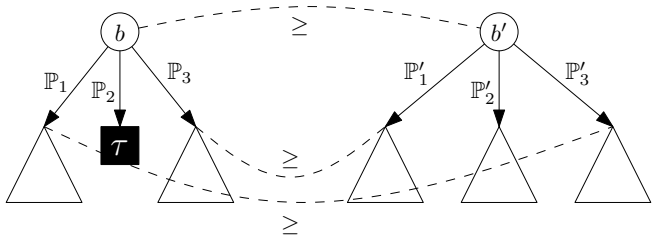

Fig. 6: The general case but now the edges are annotated with probabilities.

b' is a LOOP node (LOOPXOR, LOOPDEF) and the DO and the EXIT are the only children related of the LOOP, then b is at-least-as-good as b'. This comparison to the LOOP stems from the fact that DO is executed at least once and is eventually followed by EXIT. Thus they are in a sequence.

For choices, we need more information, i.e., we need to know the probability of executing a particular child. Therefore, we extend the general case with probabilities yielding Fig. 6. Note that implicitly the general case also contains probabilities but these are all 1, e.g., in a sequence there is no option to not execute a particular child.

Comparing two XOR/DEF nodes with each other requires that, apart from the comparison of the general case, the probabilities for the related nodes are the same. Note that in the general case, unrelated children of b are only allowed to be silent tasks, which means that these have a throughput time of 0 making them at-least-as-good as any unmapped child of b'. From this, with the requirement of equal probability between the related nodes, we know the same fraction of cases goes to unrelated nodes in both b and b'. For this fraction of cases, we know the unrelated children in b are at-least-as-good as the unrelated children in b'. Comparing two OR nodes is similar to two XOR/DEF nodes, only the probabilities of the children in b have to be at most the probabilities of the related children of b'. Note that the sum of the probabilities on the outgoing edges of an OR is at least 1. The reasoning behind this at most is that more cases having to be executed by a particular node (i.e., a higher probability) does not lower the throughput time and thus is that node at-least-as-good as the related node. This also holds when comparing a XOR/DEF with an OR, i.e., the probabilities of the children of the XOR/DEF have to be at most the probabilities of the related children in OR whilst adhering to the general case.

Next to comparing choices with each other, we can also compare choices to SEQ and AND using the earlier observation that the children of the SEQ and AND have implicitly a probability of 1. The rules are the same as for the comparison with the OR, i.e., the probabilities are at most the probabilities of the SEQ and AND, and the general case is adhered to.

We can also compare choices to LOOP nodes. For this it is sufficient to adhere to the general case and the probability of the child of b related to the REDO should be at most the probability of the REDO in b'. The intuition behind this is that the probability of the DO and EXIT are both 1, i.e., they both are executed at

Table 1: The rules the combinations of nodes have to adhere to in order to deduce that b is at-least-as-good as b'. An explanation of the used numbers can be found at the bottom.

b \ b'	SEQ	AND	EVENT	LOOP	XOR/DEF	OR
SEQ	0	-	-	1	-	-
AND	0	0	-	1	-	-
EVENT	-	-	0	-	-	-
LOOP	-	-	-	4	-	-
XOR/DEF	0	0	-	4	2	3
OR	0	0	-	4	-	3

0: general case
1: general case and only the DO and EXIT are mapped.
2: general case and the probabilities of mapped nodes are equal.
3: general case and the probabilities are at most the probability of the mapped node.
4: general case and the probability for the REDO is at most the probability of the mapped node
-: not (yet) supported.

least once. Thus we have to make sure the child related to the REDO is executed at most as often as the REDO, i.e., the probability of the node related to the REDO is at most the probability of the REDO.

Finally, in order to compare two LOOP nodes, we need to have the general case. On top of this, we need that the probability of executing the REDO of b is at most the probability of executing the REDO of b'. The idea behind this is that the higher the probability of the REDO, the more often the LOOP will be executed yielding a higher throughput time.

The requirements on the relation between two blocks are summarised in Table 1. The numbers indicate which requirements are to be adhered to in order for b to be at-least-as-good as b'. An explanation of the numbers is at the bottom of Table 1.

6 Empirical Evaluation

We have chosen an empirical evaluation over an asymptotic analysis since worst-case we still have to analyse all the possible instantiations. This comes from the fact that some models are incomparable (due to choices), and that, although some are at-least-as-good, the models have values for a KPI which are too close to each other making none of the models strictly better than another model.

For our empirical evaluation, we use the configurable process model from Fig. 2. We want to show that we can prune a significant part of the instantiation space prior to analysis. To analyse an instantiation, we simulate it at least 30

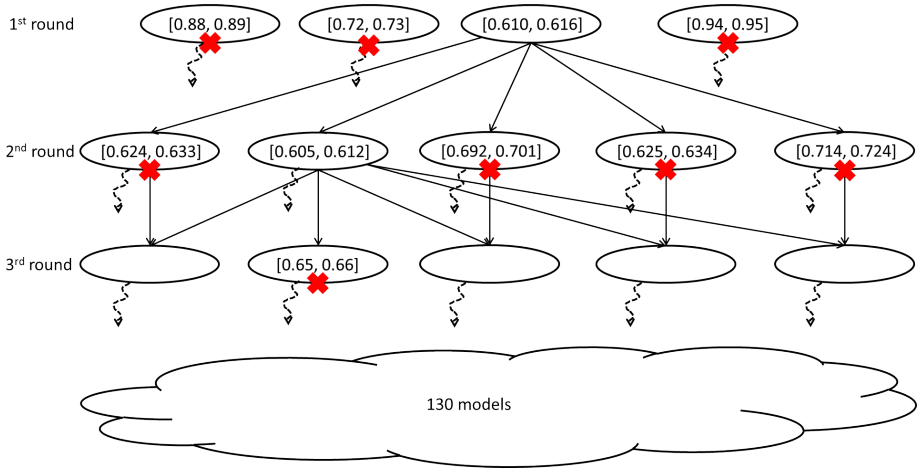

Fig. 7: The various rounds of analysis with the instantiations which were analysed and their 95% confidence intervals. We started in round 1 with the 4 most promising instantiations. In round 2, we continued with the 5 most promising instantiations from the only model that survived the first round. Finally, in round 3, we could conclude that all other instantiations are dominated.

times using L-SIM: a simulation tool developed by Lanner[3]. To enable simulation, we have extended our configurable process model with resource and timing information.

There are 144 possible instantiations from our running example (Fig. 2). Thus the collection of possible instantiations in Fig. 4 contains 144 dots. In Fig. 7, the various analysis rounds of our approach are depicted. Each round corresponds to analysing a group of instantiations which do not share an at-least-as-good relation and for which all instantiations that are at-least-as-good have already been analysed and are not dominated (yet) by another model. In the first round, we start with 4 instantiations (depicted by the 4 ovals at the top) which were most promising, i.e., there was no instantiation which was at-least-as-good as one of these 4 instantiations.

After simulating the most promising instantiations (the 95% confidence intervals of the throughput times are depicted in the ovals), only 1 of these instantiations was significantly better than the other instantiations and was kept as one of the best models. For the second round, we obtained 5 other models which were most promising (and not yet analysed) from our monotonicity. Simulating each of these models resulted in 1 model being better than the other models in the second round. This model was added to the set of best models. In the third round, again 5 models were most promising and not yet analysed. By our monotonicity notion, we knew 4 of them did not need to be analysed as non-best models from the second round were at-least-as-good as these 4. The remaining

[3] http://www.lanner.com/en/l-sim.cfm

model was simulated and was significantly worse than the best models. Since none of the models from the third round made it to the set of best models, we could conclude that all other 130 instantiations were dominated. Therefore, there was no need to analyse the other models.

From the 144 instantiations, we only had to analyse 10, which means that only 6.9% of the possible instantiations had to be analysed. Analysing all models took a bit more than 50 minutes on a single core of 2.80 GHz (including some I/O handling). The average time per model is a little bit more than 20 seconds. Computing the monotonicity of the 144 models took a bit more than 2 seconds. Only analysing the 10 models, took around 3 minutes. Note that, in Def. 1, we used the CDF for determining whether one model is at-least-as-good as another model. Since with simulation we cannot determine the CDF, we have used the confidence intervals as an approximation of this CDF.

7 Conclusions and Future Work

Within *Petra*, we analyse large amounts of instantiations from a configurable process model. These analysed instantiations are projected on a Pareto front to only keep the instantiations that are most promising, according to some Key Performance Indicators (KPIs), for an organisation. Due to the possible large amount of variation point in a configurable process model, and the resulting very large amount of possible instantiations, analysing each and every instantiation is very time consuming and unnecessary as most will never be considered by an organisation.

To prevent having to analyse all possible instantiations, using the fact that most instantiations will never be considered, we sort them according to their likelihood of appearing on the Pareto front. The sorting of the instantiations happens using our monotonicity framework. This framework can be extended to work with a multitude of KPIs.

We have applied our framework with a concrete KPI (throughput time) on the configurable process model that was used as running example, and have shown that we can achieve a significant decrease in the amount of instantiations which need to be analysed (exceeding 90%). But these results are highly dependent on the model and on the characteristics of the KPIs.

This work shows promising results and we plan to extend this into a multitude of directions. We briefly sketch a panorama of future extensions. Currently, we still need to traverse the entire instantiation space to compute the ordering between models. In the ideal case, we can constructively create the configuration for the instantiations most promising for a particular KPI. Next to this, we also want to incorporate more KPIs into the framework. Furthermore, we want to generalise this work to also be able to compute monotonicity between two process models which are not necessarily instantiations from a single configurable process model. Finally, we want to leverage some of the related work which defines monotonicity on the parameters of the configurable process model to our framework.

References

1. Schunselaar, D.M.M., Verbeek, H.M.W., Aalst, W.M.P. van der, Reijers, H.A.: Petra: Process model based Extensible Toolset for Redesign and Analysis. Technical Report BPM Center Report BPM-14-01, BPMcenter.org (2014)
2. Kung, H.T., Luccio, F., Preparata, F.P.: On finding the maxima of a set of vectors. J. ACM **22**(4) (1975) 469–476
3. Rosemann, M., Aalst, W.M.P. van der: A Configurable Reference Modelling Language. Information Systems **32**(1) (2007) 1–23
4. Gottschalk, F., van der Aalst, W.M.P., Jansen-Vullers, M., Rosa, M.L.: Configurable workflow models. International Journal on Cooperative Information Systems **17**(2) (2008) 177–221
5. Ayora, C., Torres, V., Weber, B., Reichert, M., Pelechano, V.: Vivace: A framework for the systematic evaluation of variability support in process-aware information systems. Information and Software Technology (0) (2014) –
6. Suri, R.: A concept of monotonicity and its characterization for closed queueing networks. Operations Research **33**(3) (1985) pp. 606–624
7. Mahulea, C., Recalde, L., Silva, M.: Basic server semantics and performance monotonicity of continuous petri nets. Discrete Event Dynamic Systems **19**(2) (2009) 189–212
8. Nissen, M.E.: Redesigning reengineering through measurement-driven inference. MIS Quarterly **22**(4) (1998) 509–534
9. Buzacott, J.A.: Commonalities in reengineered business processes: Models and issues. Manage. Sci. **42**(5) (May 1996) 768–782
10. van der Aalst, W.M.P.: Re-engineering knock-out processes. Decision Support Systems **30**(4) (2001) 451–468
11. Netjes, M.: Process Improvement: The Creation and Evaluation of Process. PhD thesis, Eindhoven University of Technology (2010)
12. van der Aalst, W.M.P., van Hee, K.M., ter Hofstede, A.H.M., Sidorova, N., Verbeek, H.M.W., Voorhoeve, M., Wynn, M.T.: Soundness of workflow nets: classification, decidability, and analysis. Formal Asp. Comput. **23**(3) (2011) 333–363
13. Schunselaar, D.M.M., Verbeek, H.M.W., van der Aalst, W.M.P., Reijers, H.A.: Petra: A tool for analysing a process family. In Moldt, D., Rölke, H., eds.: International Workshop on Petri Nets and Software Engineering (PNSE'14). Number 1160 in CEUR Workshop Proceedings, Aachen, CEUR-WS.org (2014) 269–288 http://ceur-ws.org/Vol-1160/.