A Structural Model Comparison for finding the Best Performing Models in a Collection

D.M.M. Schunselaar[1⋆], H.M.W. Verbeek[1⋆], H.A. Reijers[2,1⋆], 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, h.a.reijers,
w.m.p.v.d.aalst}@tue.nl
[2] VU University Amsterdam,
De Boelelaan 1105, 1081 HV Amsterdam, The Netherlands
h.a.reijers@vu.nl

Abstract. An improvement or redesign of a process often starts by modifying the model supporting the process. Analysis techniques, like simulation, can be used to evaluate alternatives. However, even a small number of design choices may lead to an explosion of models that need to be explored to find the optimal models for said process. If the exploration depends on simulation, it often becomes infeasible to simulate every model. Therefore, we define a notion of *monotonicity* to reduce the number of models required to be simulated whilst guaranteeing that the optimal models are found. We define and prove our monotonicity for throughput time. Furthermore, within our experimental evaluation we obtain very promising results in terms of running time because fewer models need to be simulated.

1 Introduction

While improving or redesigning a business process, one can model each part of the process in various ways. Even if each part has a limited number of variants, the combination of options may cause an explosion of possible models. This set of possible models, we call a *model collection*. As a redesigner is often not interested in just any model, she would like to have qualitative and quantitative information on the models in the model collection such that she can select the most suitable models, assuming one or more relevant performance criteria. In this paper, we are particulary interested in the throughput time (sometimes called flow time, sojourn time, or lead time) of a model and the best models are those models having a significant lower throughput time than the other models. Unfortunately, brute-force approaches require the simulation of each

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

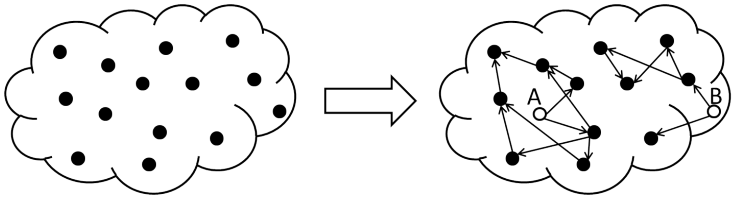

Fig. 1: Using monotonicity, we can transform the model collection on the left to the partially ordered model collection on the right. As a result, fewer alternatives need to be explored.

model, which is a time-consuming endeavour. Therefore, we present a technique to reduce the amount of models needed to be simulated, whilst guaranteeing that the best models are still found. We have chosen throughput time as this is a well-understood and often studied Key Performance Indicator (KPI) which can only be deduced from the dynamic behaviour of a model contrary to some other KPIs, e.g., number of control-tasks, which can be deduced from the structure of the model.

The aforementioned reduction is achieved by defining a *monotonicity* notion, which provides a partial order over the models. If we take the model collection on the left-hand side of Fig. 1, where each dot corresponds to a model, our particular monotonicity notion may allow for the partial order on the right-hand side of the same figure. Our monotonicity notion is based on the structure of the process models and gives a so-called *at-least-as-good* relation. If we can deduce monotonicity between a model M and a model M', then we know that the throughput time of M is at-least-as-good as the throughput time of M'. By having such a partial order, we can limit our search for the optimal models, i.e., to the models A and B in Fig. 1. We know that models not considered have poorer throughput performance. In Fig. 1, if we have simulated A and B and A is better than B, then we know that all models connected to B via the partial order do not need to be simulated as A is at-least-as-good as all of them.

In this paper, the structural comparison between two models is done using a *divide-and-conquer* approach. In this approach, each model is seen as a collection of weighted *runs* (runs are sometimes called process nets [1], or partial orders). Based on the structures of two runs, we can decide whether one run is at-least-as-good as the other run. Then, using the run weights, representing the execution likelihood, we can decide whether one model is at-least-as-good as another model.

We formally prove the correctness of our approach. Additionally, we have also implemented it to demonstrate that we indeed obtain the desired reduction in to-be simulated models and in overall analysis time. We have applied our approach on randomly generated model collections. For each of these collections, we compare the naive approach (simulate all) with the approach presented here for various sizes of subsets from the collections. On these random samples, we gain on average a reduction of 40% in the amount of models to be analysed and also a reduction of 40% in analysis time.

This paper is organised as follows: In Sect. 2, we present our definitions for runs, models, and model collections. Our monotonicity notion and proofs are presented in

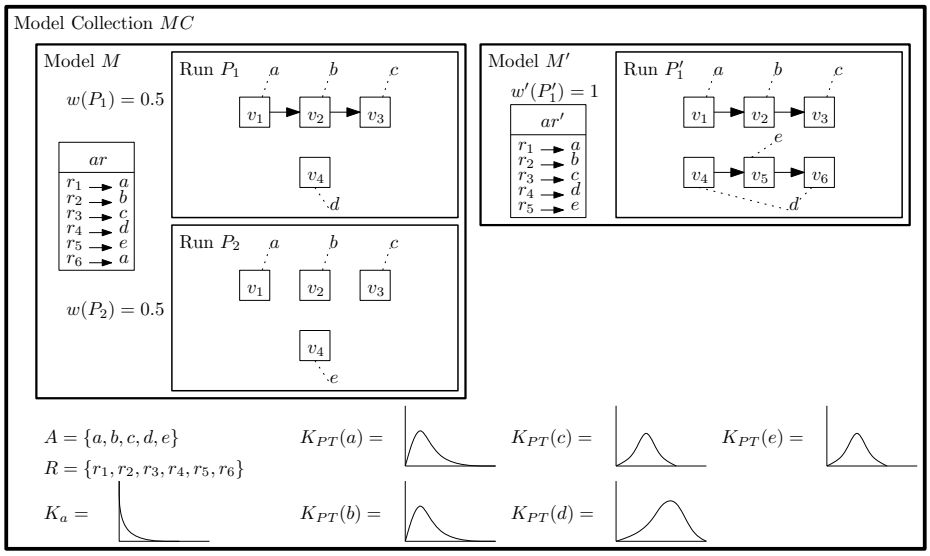

Fig. 2: An example model collection.

Sect. 3. Section 4, discusses experimental results. Finally, we conclude our paper with related work (Sect. 5) and the conclusions (Sect. 6).

2 Model Collection

In this paper, a model collection consists of models, while a model itself consists of weighted runs [1] (see Fig. 2). A run specifies the partial order of activities needed to be executed for a particular case and does not contain any choices. A run consists of *vertices* which are labelled with activities. In Fig. 2, we have a run P_1 with a vertex v_1 labelled with activity a. Next to this, a run specifies the causal relationship between the vertices by means of edges. The edge between the vertices v_4 and v_5 in run P_1' means that e can only start once vertex v_4 executing d has been completed. Prior to formally defining the runs, we introduce the set of activities, denoted by A, and the set of resources, denoted by R, which are assumed to be constant for the model collection (although it is not necessary that every activity or resource is present in each model/run). We have chosen this representation for our models as this fits better with our divide-and-conquer technique. In [1], an algorithm is presented to transform a Petri net into a collection of runs. By limiting the number of times an iteration is executed, it is possible to obtain a finite set of runs.

Definition 1 (Run). *Let A be a set of activities, then a run P over A is a 3-tuple (V, E, ℓ) where:*

- *V is a set of vertices;*
- *$E \subseteq V \times V$ is a set of directed edges;*

- $\ell \in V \rightarrow A$ *is a labelling function labelling vertices with activities;*
- *the graph* $G = (V, E)$ *is a Directed Acyclic Graph (DAG).*

The edges in a run indicate direct succession of vertices; E^+ is the partial order of the vertices.

A model consists of runs and these runs combined define the behaviour of a model. As not every run must be equally probable, we define a weight function within the model. Furthermore, a model specifies which resources can execute which activity, e.g., in model M in Fig. 2, activity a can be executed by resources r_1 and r_6. In the definition of a model, we use $\mathcal{P}_A$ as the universe of runs over A.

Definition 2 (Model). *Let* A *be a set of activities, let* R *be a set of resources, let* $\mathcal{P}_A$ *be the universe of runs over* A*, then a model* M *over* A *and* R *is a 3-tuple* (PP, w, ar) *where:*

- $PP \subseteq \mathcal{P}_A$ *is the set of runs over* A*;*
- $w \in PP \rightarrow (0, 1]$ *is a weight function indicating the probability of each run such that* $\left(\sum_{P \in PP} w(P)\right) = 1$*. Note that runs with weight 0 are not allowed;*
- $ar \in R \rightarrow A$ *indicates per resource which activity it can execute.*

Note that a resource can execute only one activity, but multiple resources can execute the same activity. We require that a resource can only perform a single activity to guarantee that we can compare the resource utilisation of both models and thus the queue time per activity based on the structure. This is due to the fact that even a small increase in the resource utilisation can have a significant effect on the throughput time. We acknowledge that this is a strong assumption, but it is purely for an end to end approach, i.e., if there is information that the queue times in one model are at-most that of the other model, then we can lift this assumption and our proofs on run and model level still hold. For our definition of our model collection, we use the universe of random variables ($\mathcal{K}$) to annotate activities with stochastic information and to be able to explicitly define the inter-arrival time of new cases. A random variable can be seen as a function giving probabilities to possible outcomes, e.g., the probability that a coin will give heads. Both the stochastic information of activities and the inter-arrival time of new cases are on the collection level and thus are the same for all models in the collection. Next to this, we use $\mathcal{M}_{A,R}$ as the universe of models over A and R.

Definition 3 (Model Collection). *Let* $\mathcal{K}$ *be the universe of random variables, then a model collection* MC *is a 5-tuple* (A, R, MM, K_a, K_{PT}) *where:*

- A *is the set of activities;*
- R *is the set of resources;*
- $MM \subseteq \mathcal{M}_{A,R}$ *is the collection of models over* A *and* R*;*
- $K_a \in \mathcal{K}$ *is the random variable describing the inter-arrival time of new cases;*
- $K_{PT} \in A \rightarrow \mathcal{K}$ *maps every activity onto the random variable describing the duration of that activity.*

Note that our model collection is less general than most other model collections as our model collection is a body of scenarios for executing the same process.

3 Throughput Time

As mentioned before, we focus on the throughput time. The throughput time of a single case is the time between arrival of this case and the moment the case is finished. The throughput time of a run P in a model M in a model collection MC for an arbitrary case is described by the random variable $K_{TT}(MC, M, P) \in \mathcal{K}$. We consider the situation where the model and runs are in steady state. This will be achieved if the utilisation of the resources is below 1.0 because K_a is independent.

Definition 4 (Throughput time of a run). *Let $MC = (A, R, MM, K_a, K_{PT})$ be a model collection, let $M = (PP, w, ar)$ be a model in MM, let $P = (V, E, \ell)$ be a run in PP, then the throughput time of P in steady state is the random variable $K_{TT}(MC, M, P) \in \mathcal{K}$.*

We are interested in the model with the lowest throughput time which requires the comparison of two random variables. For this comparison, we use the Cumulative Distribution Function (CDF) of the throughput time which we also use as our notion of throughput time KPI. Note that our approach is not limited to this definition of the throughput time KPI. Any definition works as long as it is monotone, i.e., if the throughput time increases, then the KPI should decrease.

Definition 5 (Throughput time KPI). *Let K_{TT} be the random variable describing the throughput time, let $\mathbb{P}(K_{TT} \leq x)$ be the probability that K_{TT} is at-most x, then the throughput time KPI, denoted by F_{TT}, is defined as: $\forall_x(F_{TT}(x) = \mathbb{P}(K_{TT} \leq x))$.*

The throughput time KPI of a run is denoted by $F_{TT}(MC, M, P)$. Since our models consist of a collection of runs with a weight function, we define the throughput time of a model as the weighted sum of the throughput times of the runs.

Definition 6 (Throughput time of a model). *Let $MC = (A, R, MM, K_a, K_{PT})$ be a model collection, let $M = (PP, w, ar)$ be a model in MM, then the throughput time of M in steady state is the random variable $K_{TT}(MC, M) \in \mathcal{K}$, which is defined as follows:*

$$K_{TT}(MC, M) = \sum_{P \in PP} w(P) \cdot K_{TT}(MC, M, P)$$

The throughput time KPI of a model is denoted by $F_{TT}(MC, M)$. Having the throughput time KPI of a model in place, we can now define the at-least-as-good relation between two models.

Definition 7 (At-least-as-good models). *Let $MC = (A, R, MM, K_a, K_{PT})$ be a model collection, and let M, M' be two models from MM, we say M is at-least-as-good as M', denoted by $M \geq M'$ if $\forall_x(F_{TT}(MC, M)(x) \geq F_{TT}(MC, M')\ (x))$.*

By having that the throughput time KPI of M is above that of M', i.e., the probability that it stays below a certain point x is greater, we guarantee that in general M has a lower throughput time. It is, however, still possible that for an individual case the throughput time of M' is better than M. We are, however, interested in an overall comparison.

Taking the models from the collection in Fig. 2, Fig. 3 shows the possible throughput time KPIs for the models. By having that the KPI of M is above M', we conclude that M is at-least-as-good as M'.

Having our definitions of throughput time on run and model level and the corresponding throughput time KPIs, we define the at-least-as-good relation between runs and show how this can be deduced based on the structures of the runs. Afterwards, we show how the at-least-as-good relation between runs can be leveraged to the model to provide the at-least-as-good relation.

3.1 At-least-as-good runs

Prior to giving the structural requirements on two runs for the at-least-as-good relation, we shall define first when a run is at-least-as-good as another run in terms of the throughput time KPI.

Definition 8 (At-least-as-good runs). *Let $MC = (A, R, MM, K_a, K_{PT})$ be a model collection, let $M = (PP, w, ar)$, $M' = (PP', w', ar')$ be two models from MM, and let P, P' be two runs from PP and PP' respectively, then we say P is at-least-as-good as P', denoted by $P \geq P'$, if $\forall_x (F_{TT}(MC, M, P)(x) \geq F_{TT}(MC, M', P')(x))$.*

Informally when comparing the structures of the runs P and P', if P has fewer work, or has more flexibility in the order of executing activities, then P can do things faster than P' (assuming M has not less resources per activity). We operationalise this by comparing the vertices in P and P' (fewer vertices is fewer work), and by comparing the edges (flexibility in the ordering). Next to this, we need to make the requirement that we only compare vertices with the same label. When comparing two runs, we abstract from the respective models these runs are part of. In the comparison of two models, we shall elaborate on this.

Since which resource can execute which activity is defined on model level, we first introduce when a model M is at-least-as-good as M' with respect to the resource allocation, denoted by $M \geq_{ar} M'$.

Definition 9 (Structurally at-least-as-good resource allocation). *Let $MC = (A, R, MM, K_a, K_{PT})$ be a model collection, let $M = (PP, w, ar)$, $M' = (PP', w', ar')$ be two models from MM, then we say M is at-least-as-good as M' with respect to the resource allocation, denoted by $M \geq_{ar} M'$, if $dom(ar') \subseteq dom(ar) \wedge \forall_{r \in dom(ar')} ar(r) = ar'(r)$, i.e., every activity in M can be performed by at least the same resources than that activity in M'.*

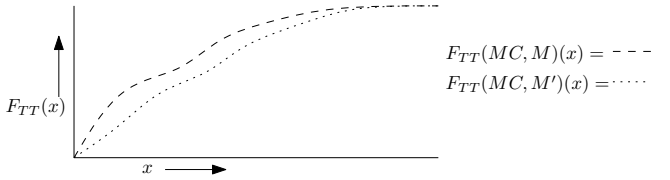

Fig. 3: Example throughput time KPIs for the models from Fig. 2.

Definition 10 (Structurally at-least-as-good runs). *Let $MC = (A, R, MM, K_a, K_{PT})$ be a model collection, let $M = (PP, w, ar)$, $M' = (PP', w', ar')$ be two models from MM such that $M \geq_{ar} M'$, let $P = (V, E, \ell)$, $P' = (V', E', \ell')$ be two runs from PP and PP', and let $map \in V \to V'$ be an injective mapping from vertices in P to vertices of P', then we say P is structurally at-least-as-good as P' given mapping map (denoted by $P \geq_s^{map} P'$) if and only if:*

1. *$\{map(v) \mid v \in V\} \subseteq V'$, every vertex in P is mapped onto some vertex in P';*
2. *$\{(map(u), map(v)) \mid (u, v) \in E^+\} \subseteq E'^+$, every path in P is mapped to some path in P' (where E^+ and E'^+ are the partial orders of the vertices);*
3. *$\forall_{v \in V} \ell(v) = \ell'(map(v))$, every vertex in P is mapped onto a vertex in P' labelled with the same activity.*

We say P is structurally at-least-as-good as P' (denoted by $P \geq_s P'$), if a mapping $map : V \to V'$ exists such that $P \geq_s^{map} P'$.

For Def. 10, we take two runs and compute the partial order, e.g., if we take P_1 and P_1' from Fig. 2, we obtain the partial orders in Fig. 4 (we have given the vertices from P_1' different names). Between these partial orders, we create a mapping which does not need to be unique, e.g., v_4 could also have been mapped onto v_6'. Taking also ar and ar' from Fig. 2 into account, we say P_1 is structurally at-least-as-good as P_1' since there exists a mapping such that: (a) each vertex in P_1 is mapped onto a vertex in P_1', (b) P_1 has fewer edges in the partial order, and (c) on model level, a can be executed by r_1 and r_6.

We first prove that the throughput time per vertex in a run P is at-least-as-good as the throughput time of the vertex on which it is mapped in a run P'. Afterwards, we prove that if the throughput times per vertex are at-least-as-good, then also P is at-least-as-good as P'. The throughput time of a vertex v consists of the *queue time* and the *processing time*. Queue time is the time between the moment that a work item arrives at v and the moment the resource *starts* working it. The processing time is the time between when a resource *starts* working on a particular work item at v and the moment it is *finished*. We denote the throughput time KPI of a vertex by: $F_{TT}(MC, M, P, v)$.

One vertex is at-least-as-good as another vertex if the throughput time KPI of the former is at-least-as-good as the latter.

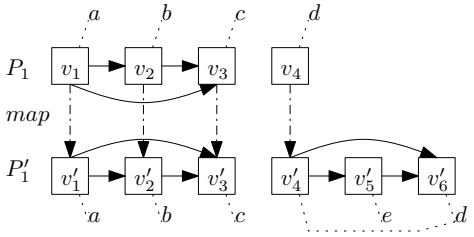

Fig. 4: The partial orders of the runs P and P' with a mapping map such that $P \geq_s^{map} P'$. Note that a mapping where v_4 is mapped onto v_6' would also have been fine.

Definition 11 (At-least-as-good vertices). *Let $MC = (A, R, MM, K_a, K_{PT})$ be a model collection, let $M = (PP, w, ar)$, $M' = (PP', w', ar')$ be two models from MM, let P, P' be two runs from PP, PP' respectively, and let v, v' be two vertices from V and V', we say v is at-least-as-good as v', denoted by $v \geq v'$ if $\forall_x(F_{TT}(MC, M, P, v)(x) \geq F_{TT}(MC, M', P', v')(x))$.*

Prior to giving the proof that the vertices in P are at-least-as-good as the vertices they are mapped on in P', we first have to make the following assumptions:

Assumption set 1

1. The amount of arriving cases per time unit is exactly the same per run;
2. There is a single First In First Out (FIFO) queue per activity from which resources execute work items. This FIFO queue contains all the work items currently in the queues of the vertices labelled with a particular activity;
3. Having more resources for an activity cannot increase the queue time of the FIFO queue for that activity if the amount of cases per time unit stays exactly the same.

Theorem 1. *Let $MC = (A, R, MM, K_a, K_{PT})$ be a model collection, let $M = (PP, w, ar)$, $M' = (PP', w', ar')$ be two models from MM such that $M \geq_{ar} M'$, let $P = (V, E, \ell)$, $P' = (V', E', \ell')$ be two runs from PP, PP' respectively, let map be a mapping such that $P \geq_s^{map} P'$, and let v, v' be two vertices from V and V' such that $map(v) = v'$. If the assumptions in Assumption set 1 hold, then we have $\forall_x(F_{TT}(MC, M, P, v)(x) \geq F_{TT}(MC, M', P', v')(x))$.*

Proof. Recall that the throughput time of a vertex consists of queue time and processing time. By K_{PT}'s nature and the fact that v and v' have the same label, we know that their processing times are the same. Now we only need to focus on the queueing time per vertex. We shall do this via the FIFO queue per activity. From Def. 9, we know that every activity in P has at-least the same resources as in P'. Furthermore, Def. 10 sub 1 gives us together with Def. 10 sub 3 that, for a particular activity, P has at-most the same amount of vertices labelled with this activity as P'. This, together with assumptions 1 and 3, yields that the queue time per activity in P is at-most that in P'. From assumption 2, we can conclude that the queue time of v is at-least-as-good as the queue time of v' as the queue time per vertex is the queue time per activity. Combining that the queue time and the processing time are both at-least-as-good, we conclude that $\forall_x(F_{TT}(MC, M, P, v)(x) \geq F_{TT}(MC, M', P', v')(x))$. □

Assumptions 1 and 3 in Assumption set 1 allow us to compare the queue times for a particular activity. This is not yet enough for comparing the queue times between v and v', e.g., it might be that the queue time on activity level is smaller, but, at a particular vertex, it could have increased. By having the FIFO queue, we prevent this from happening.

Having shown that the vertices in P are at-least-as-good as the vertices they are mapped on in P', we can now prove that P is at-least-as-good as P'.

Theorem 2. *Let $MC = (A, R, MM, K_a, K_{PT})$ be a model collection, let $M = (PP, w, ar)$ and $M' = (PP', w', ar')$ be two models from MM such that $M \geq_{ar} M'$, let*

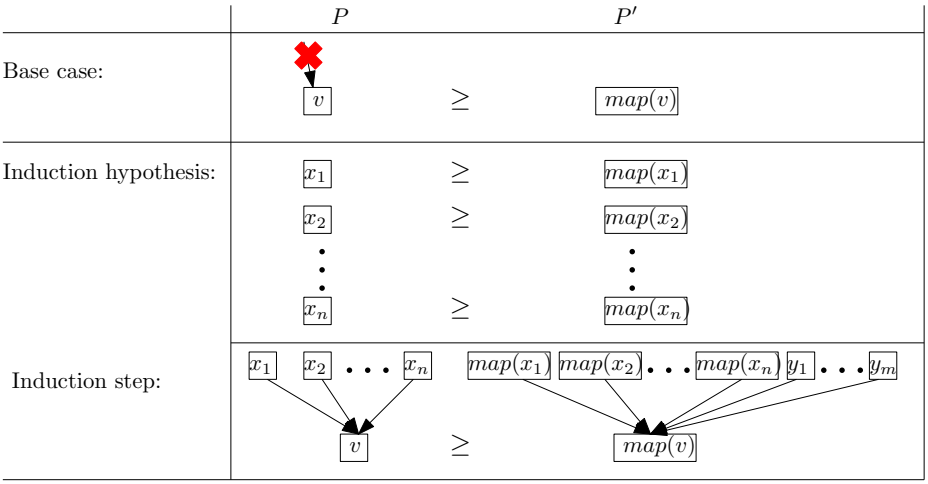

Fig. 5: Supporting figure for our proof of $P \geq P'$.

$P = (V, E, \ell)$ be a run of PP, let $P' = (V', E', \ell')$ be a run of PP′, and let map be a mapping such that $P \geq_s^{map} P'$. If the assumptions in Assumption set 1 hold, then we have $\forall_x(F_{TT}(MC, M, P)(x) \geq F_{TT}(MC, M', P')(x))$.

Proof. We shall prove this by induction on the structure of the runs (see also Fig. 5) which we can do since our runs are DAGs. For this, we look at the throughput time up-to and including a particular vertex. In the base case, we take the vertices from P which do *not* have any incoming edges. For each vertex v, we know there is a vertex $map(v)$ and from Thm. 1, we know the throughput time of v is at-least-as-good as that of $map(v)$. In P', $map(v)$ can have incoming edges but incoming edges cannot decrease the throughput time. Hence we have that the throughput time up-to and including v is at-least-as-good as $map(v)$.

For the induction hypothesis, we assume that at least the vertices $X = \{x_1, \ldots, x_n\}$ are at-least-as-good as their mapped vertices. What remains to prove is that given a v such that the source of all its incoming edges are in X, i.e., $\forall(u, v) \in E^+ : u \in X$, we can conclude that $v \geq map(v)$. For this it is important to see that the throughput time up-to and including a vertex is the maximum over its incoming edges plus the throughput time of the vertex itself. Furthermore, from Def. 10 sub 2, we know that if there is an edge between two vertices in P, it is also in P'. This means that $\forall x \in X : (map(x), map(v)) \in {E'}^+$. Finally, it might be that $map(v)$ has more incoming edges, e.g., $y_1, \ldots, y_m$ in Fig. 5. Given the fact that the throughput time of a vertex is the maximum over its incoming edges plus the throughput time of the vertex itself, we can conclude that the maximum over the incoming edges of v is at-most that of $map(v)$ and that the vertices $y_1, \ldots, y_m$ cannot decrease the maximum as it is a monotone function. Using Thm. 1, we also know that v is at-least-as-good as $map(v)$. Hence the throughput time up-to and including v is at-least-as-good as $map(v)$.

By having deduced that for all vertices in P the throughput time up-to and including that vertex is at-least-as-good as their mapped vertex in P', we can conclude that $\forall_x(F_{TT}(MC, M, P)(x) \geq F_{TT}(MC, M', P')\ (x))$ as the throughput time of a run is the maximum over all throughput times up-to and including the vertices. □

After showing the correctness of the at-least-as-good relation between runs, we now define our at-least-as-good relation between models. The at-least-as-good relation between models holds if we can find a valid *matching graph* between the runs of the models. Graphically, a matching graph can be seen as a bipartite graph (Fig. 6). The runs of M are on the left-hand side and the runs of M' are on the right-hand side together with their weights. An edge between two runs indicates that the run on the left-hand side is at-least-as-good as the run on the right-hand side, e.g., P_1 is at-least-as-good as P_1'. As the weight of a run gives the probability of this run occurring it also gives the fraction of cases arriving for this run, we need to take this into account in the matching graph due to Thm. 2. Therefore, we have weights on the edges in the matching graph indicating the weight of the runs when they are compared. For instance, the 0.5 between P_2 and P_1' indicates that in the comparison of P_2 and P_1', we give them both a weight of 0.5.

As the weights on the edges in the matching graph indicate the weight of the runs when they are compared, we need to guarantee that the sum of the weights on the outgoing edges of a run is always that of the actual weight of that run. The same holds for the weights of the incoming edges of a run. The weights in the matching graph should be in $[0, 1]$. A (valid) matching graph is defined as:

Definition 12 ((Valid) Matching Graph). *Let $MC = (A, R, MM, K_a, K_{PT})$ be a model collection, and let $M = (PP, w, ar)$ and $M' = (PP', w', ar')$ be two models from MM such that $M \geq_{ar} M'$, then the matching graph, between M and M', denoted by $match_{M,M'}$, is defined as: $match_{M,M'} \in (PP \times PP') \rightarrow [0, 1]$. We say $match_{M,M'}$ is valid if and only if:*

- *$\forall (P, P') \in dom(match_{M,M'}) : P \geq_s P'$, if there is an edge between two runs in the matching graph, then the first is structurally at-least-as-good as the latter;*
- *$\forall P \in PP : w(P) = \sum(P, P') \in dom(match_{M,M'}) : match_{M,M'}(P, P')$, the weights of the outgoing edges are the same as the weight of the run;*
- *$\forall P' \in PP' : w'(P') = \sum(P, P') \in dom(match_{M,M'}) : match_{M,M'}(P, P')$, the weights of the incoming edges are the same as the weight of the run.*

We can now prove the following:

Theorem 3. *Let $MC = (A, R, MM, K_a, K_{PT})$ be a model collection, let $M = (PP, w, ar)$, $M' = (PP', w', ar')$ be two models from MM, then if we have a valid matching graph between M and M', it follows that $\forall_x(F_{TT}(MC, M)(x) \geq F_{TT}(MC, M')(x))$.*

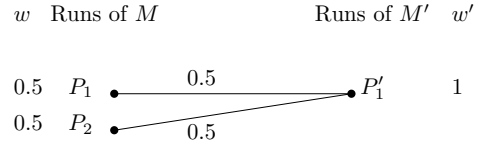

Fig. 6: An example valid matching graph.

Proof. The throughput time KPI of a model is defined on the throughput time of a model which in turn is built-up from the throughput times of the individual runs. From our valid matching graph, we obtain that, disregarding that resources are shared between runs, M is at-least-as-good as M'. This follows from the fact that for every run P in PP, there is a collection of runs in PP' to which P is at-least-as-good and the weight of P is the same as the weight of the collection of runs (using the edges in the matching graph). By having equal weights, we fulfill the first assumption in Assumption set 1 as the weight of a run is directly linked to the amount of arriving cases per time unit.

We now only have to pay attention to the fact that resources are shared amongst runs which has an effect on the queue times. From Thm. 1, we obtain that the queue time per activity for a run P is at-least-as-good as the queue time of that activity in the run P' for which it holds that $P \geq_s P'$ (under an equal amount of arriving cases per time unit). As this holds under any amount of arriving cases (as long as the model as a whole stays in steady state), this also holds when K_a amount of cases arrive. Using the edges in the valid matching graph, we can distribute the K_a amongst the runs. By taking a single queue per activity for the whole model, we can combine the queues per run into this and we know the queue time of an activity in M is at-least-as-good as the queue time of that activity in M'. Hence we can conclude that: $\forall_x(F_{TT}(MC, M)(x) \geq F_{TT}(MC, M')(x))$. □

Using the matching graph, it becomes possible to structurally compare models with each other. If we are able to obtain a valid matching graph, we can conclude that one model is at-least-as-good as another model. In the next section, we apply our technique to show the gains it may bring.

4 Experimentation

For our experimentation, we have randomly generated 30 model collections varying in size (16 to 12288 models)[3]. From these model collections, we have randomly selected a subset of the models within a collection. On these subsets, we apply our monotonicity approach and compare this to naively simulating each model in this collection. Our goal is to find the models which have a significantly lower throughput time than the other models while spending less time on simulations. In our explorative experiment, we use *Petra* [2] in conjunction with the simulation engine L-SIM[4], which is BPSim [3] compliant. We have chosen to generate the model collections ourselves as real-world model collections lack simulation data. Note that by simulating the models we obtain an approximation of the mean value of a KPI and not the exact probability distribution.

4.1 Generating the model collection

To create a model collection, we randomly generate a single model as a basis. The base model is generated according to the following characteristics: The number of activities

[3] The data and results can be downloaded from https://svn.win.tue.nl/repos/prom/Documentation/Petra/BPM2015SchunselaarCaseStudyData.7z.

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

is normally distributed with mean 20 and variance 4; The number of resources per activity is normally distributed with mean 7 and variance 0.5.

Having generated the activities, we create bottom-up a block-structured process model out of the activities. We first randomly select a number of activities (normally distributed with mean 5 and variance 1) and place these activities in one of the following relations: sequence, parallel, choice (exclusive and inclusive), or in an iteration. All of these relations have the same probability of being selected. Afterwards, we take the remaining activities and the newly added relation and again select a relation to hold. We keep on doing this until all activities are related.

For our model collection, we have the following characteristics: All processing times for activities (K_{PT}) are normally distributed with a mean of 10 minutes and a variance of 1 minute; The inter-arrival time of new cases (K_a) is negative exponentially distributed with a mean of 15 minutes.

Each model within the collection is subsequently obtained by randomly removing parts of the base model. Prior to adding the newly created model to the collection, we verify that the model ends up in steady state by doing a simulation to check the development of the utilisation rates. If the model ends up in steady state, it is added, otherwise it is discarded.

4.2 Analysis of a model collection using monotonicity and naively

In the naive case, all models from the actual subset are simulated to obtain estimated throughput times. We propose to first determine the monotonicity relation between the models in a collection and then in the second stage, only to simulate the most promising models until all non-simulated models are dominated by the simulated models. In order to determine the monotonicity relation, we first decompose the process model into the various runs. For this, we unroll the iterations 0 to 2 times.

For each pair of runs, we create the mapping (if it exists) in such a way that the first run is structurally at-least-as-good as the second run. Since the computation of this mapping is quite expensive in terms of running time, our implementation employs a heuristic to balance between the time needed for simulation and the time needed for computing the monotonicity, i.e., if we expect that computing the monotonicity between two models will take more time than simulating the individual models, then we prefer to just run the simulation. In previous experiments, we have determined that 10,000 for the product of the number of runs is a good value for the heuristic, e.g., if one model allows for 1,000 runs and the other model for 12, then the product is 12,000 and the monotonicity will not be computed. As a result, we would have no information whether the one model is at-least-as-good as the other or vice versa.

The analysis is done for increasing sizes of subsets and earlier subsets are maintained, i.e., models are only added. To limit the variability in results for the same model, we only simulate each model once and use these outcomes throughout the experiment.

4.3 Results

We gain on average a reduction of 40% in total time to find the optimal models (Fig. 7). Also interesting to note is that the overhead is around 3%, i.e., if the monotonicity is

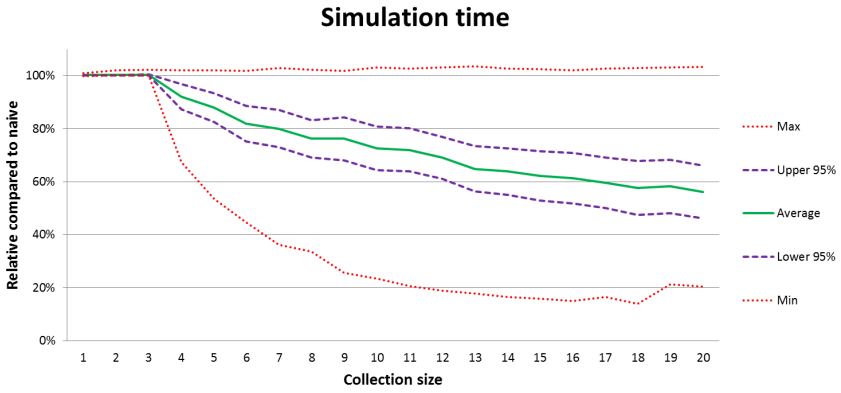

Fig. 7: Relative quantitative comparison. 100% is the time of the naive approach.

not reducing the amount of models which need to be analysed, this does not incur a heavy computational penalty also due to our earlier introduced heuristic. The reduction of models which need to be inspected is similar to the reduction in simulation time. Note that the minimum and maximum do not belong to a single model collection but instead it is minimum and maximum over all model collections.

We can see in Fig. 8 that only 8 out of 20 models had to be simulated to obtain the optimal models for model collection 2. Each model that has a cross means that there was a model at-least-as-good as this model and that model was not optimal.

The models 1, 4, and 6 from model collection 2 are depicted in Fig. 9 using the BPMN formalism. The models 4 and 6 can be obtained by removing the indicated part. Our monotonicity notion concludes that model 4 is at-least-as-good as 6. After simulating model 4 (which is not optimal), we know model 6 cannot be better and hence is not simulated. We cannot conclude monotonicity between model 4 and model 1 because of the difference in the choices they can make which gives a different weighing to the runs and hence both need to be analysed.

4.4 Threats to validity

A threat to the validity of our experiments is the fact that we simulate the models beforehand and use the results and running times of them. By doing so, we gain a higher similarity in the output of the monotonicity and naive implementation, i.e., the same models are considered the best model. As such, we can better compare the results from both approaches. But at the same time, we introduce look-up time necessary for loading the analysed model in main memory which is beneficial for our approach. We argue that the overhead of this is insignificant as the simulation itself is at least 3 orders of magnitude larger.

Another threat to the validity of the results lies in our assumptions on the models. The assumptions that a resource can only perform a single activity and the fact that we have dedicated resources to the model are strict, but without these we can have counterintuitive results. For instance, having a resource 100% dedicated to a process

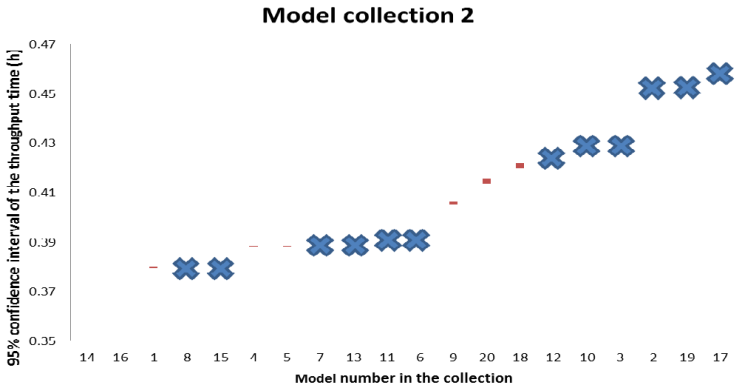

Fig. 8: Analysis results for model collection 2. The crosses indicate models which did not have to be analysed thanks to monotonicity.

can be significantly different than having two resources each 50% dedicated to the process [4]. Furthermore, having a resource only available on Monday might even favour more edges in the run as this might limit the possibility that workitems become available after Monday. By having fewer edges, the variance of the throughput time can increase resulting in false positives.

5 Related Work

By analysing redesign alternative encapsulated in a model collection, our work can be positioned on the intersection of *model collections*, and *performance evaluation*. We first discuss work from each of the two areas and then discuss work on the intersection.

In [5], the authors list the research areas within model collections. Often these model collections lack sufficient information for quantitative analysis, i.e., the context is miss-

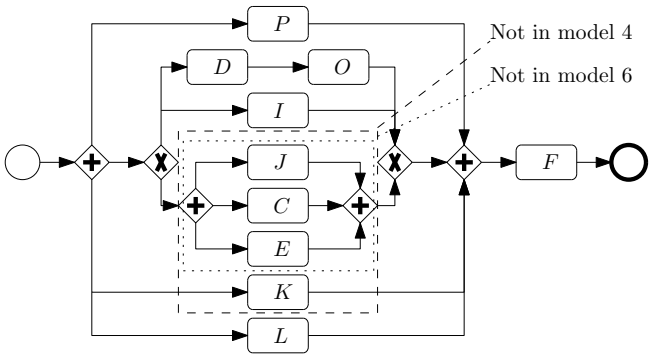

Fig. 9: The models 1, 4, and 6 from model collection 2.

ing, e.g., the arrival process of new cases, duration of activities, etc. If the model collection is viewed from a specific context, then our technique can be most beneficial in querying the collection of models. For instance, in PQL [6], the user can specify she is interested in models where an activity A is eventually followed by an activity B. As there might be a large amount of models returned from a query, our technique can be used to structurally order these models based on the throughput time. In this way, the user is immediately presented with the most promising models whilst adhering to the earlier specified structural requirements.

Within performance evaluation, the idea of monotonicity is not new and in queueing theory it has already been pursued [7]. In [7], the notion of monotonicity is similar but they focus on the parameters of the network and not the topology of the network. The work in [8] is similar to the work in [7] but now defined on continuous Petri nets. Since runs can be translated to Petri nets, this might be an interesting approach to use in the at-least-as-good relation between runs.

In [9], an approach is presented to evaluate when certain changes to the structure of the process model are appropriate. Starting from commonalities in reengineered processes, the paper deduces under which circumstances a change to the structure of the model is beneficial. The majority of the authors' ideas is not tailored towards throughput time but some ideas can be applied to our setting. These ideas are mainly on how resources perform their tasks.

So-called Knock-Out systems are discussed in [10] and heuristics for optimising these are defined. A Knock-Out system is a process model where after each task or group of tasks in case they are in parallel a decision is made to continue with the process or to terminate. The goal is to rearrange the tasks in such a way that the resource utilisation and flow time (throughput time) are optimised whilst adhering to constraints on the order of tasks encoded in precedence relations. By having an approach starting from a single model, this approach is not directly applicable to comparing two models.

In [11], a tool called KOPeR (Knowledgebased Organizational Process Redesign) is presented. KOPeR starts from a single model and identifies redesign possibilities. These redesign possibilities are simulated to obtain performance characteristics. This approach is not tailored towards directly comparing two models to determine which is at-least-as-good but our approach can be used to discard models prior to simulation.

In [12], process alternatives are analysed which have been obtained by applying redesign principles. Similar to the work in [11], our approach can aid in reducing the amount of to-be-analysed redesign options.

In our previous work, we have presented *Petra* a toolset for analysing a family of process models [13]. A family of process models is similar to a model collection but models are closer related. The work here can improve *Petra* by a-priori sorting the process models and only analyse the models most promising.

6 Conclusion

We have shown an approach to structurally compare the models within a model collection resulting in an at-least-as-good relation between models based on throughput time. This at-least-as-good relation can be used to minimise the effort to simulate a collection

of highly similar models. This is particularly useful if redesigning an existing process where different improvement opportunities exist. Our approach poses a number of restrictions on the resources. In particular, we demand that resources can only execute a single activity and that they are truly dedicated to the process in question.

Next to showing the theoretical validity our approach, we also have applied our approach in an experimental setting. Within this experiment, we have generated our own model collections and demonstrated the gains. We gained on average a reduction of more than 40% of the models which no longer needed to be simulated. Furthermore, we obtained a reduction of 40% in the time it takes to find the optimal models. Using our technique, simulation becomes a much more feasible approach, supporting a more quantitative approach to process redesign.

For future work, an interesting question is which of our assumptions can be relaxed to allow for the inclusion of a wider set of models to be considered. In particular, we want to look into whether runs have to be directed or whether they are also allowed to be undirected. This would allow us to compare different sequences of tasks and greatly increase our applicability.

References

1. Desel, J.: Validation of Process Models by Construction of Process Nets. In: BPM. Volume 1806 of Lecture Notes in Computer Science., Springer (2000) 110–128
2. 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)
3. Gagne, D., Shapiro, R.: BPSim 1.0. http://bpsim.org/specifications/1.0/WFMC-BPSWG-2012-01.pdf (Feb 2013)
4. Aalst, W.M.P. van der, Nakatumba, J., Rozinat, A., Russell, N.: Business Process Simulation. In: Handbook on Business Process Management 1. International Handbooks on Information Systems. Springer Berlin Heidelberg (2010) 313–338
5. Dijkman, R.M., La Rosa, M., Reijers, H.A.: Managing Large Collections of Business Process Models - Current techniques and challenges. Computers in Industry **63**(2) (2012) 91–97
6. Hofstede, A.H.M. ter, Ouyang, C., La Rosa, M., Song, L., Wang, J., Polyvyanyy, A.: APQL: A Process-Model Query Language. In Song, M., Wynn, M.T., Liu, J., eds.: AP-BPM 2013. Volume 159 of Lecture Notes in Business Information Processing., Springer (2013) 23–38
7. Suri, R.: A Concept of Monotonicity and Its Characterization for Closed Queueing Networks. Operations Research **33**(3) (1985) pp. 606–624
8. 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
9. Buzacott, J.A.: Commonalities in Reengineered Business Processes: Models and Issues. Manage. Sci. **42**(5) (May 1996) 768–782
10. Aalst, W.M.P. van der: Re-engineering Knock-out Processes. Decision Support Systems **30**(4) (2001) 451–468
11. Nissen, M.E.: Redesigning Reengineering Through Measurement-Driven Inference. MIS Quarterly **22**(4) (1998) 509–534
12. Netjes, M.: Process Improvement: The Creation and Evaluation of Process. PhD thesis, Eindhoven University of Technology (2010)
13. Schunselaar, D.M.M., Verbeek, H.M.W., Aalst, W.M.P. van der, Reijers, H.A.: Petra: A Tool for Analysing a Process Family. In: PNSE'14. Number 1160 in CEUR Workshop Proceedings, CEUR-WS.org (2014) 269–288