Supplementary Materials: Experiments

We have analyzed data-aware soundness verification algorithm and determined parameters influencing the complexity of this algorithm. Such parameters include the number of places, the number of transitions, the number of arcs, the number of variables and the number of atomic conditions. In Chapter 1, we describe the developed tool that has been used to generate sample DPNs according to these predefined parameters. In Chapter 2, we demonstrate the results of experiments conducted with the help of this tool. The source Jupyter notebook with experiments results is available at Github[1].

[1] https://github.com/NIRS-Supplementary/Experiments-source/tree/main

1 Tool for generating DPNs with predefined parameters

In this chapter, we describe the instruments that have been developed for performance evaluation of the soundness verification algorithm. Section 1.1 explains the approach to generate DPNs with predefined parameters and describes the class library that implements this approach. Section 1.2 introduces a tool that allows to automate the process of generation and verification of DPNs.

1.1 Generation of DPNs with predefined parameters

We have separated the task of generating a DPN with the predefined parameters on the following subtasks:

1. Generation of a sound backbone for a DPN.
2. Addition of extra arcs to the net.
3. Addition of variables and conditions to the net.

Generation of a sound backbone allows to construct a sound Petri net according to the predefined number of places and transitions. The generated net has one input place and one output place. Cycles inside the net are prohibited. Weight of each arc in the net is equal to 1. The number of places in a preset and a postset of each transition is equal to 1. Each place besides an input place has an input arc, and each place besides an output place has an output arc. By following the mentioned properties, we ensure that the result model is sound.

At the second step, we add extra arcs to the generated sound Petri net. The user defines the number of extra arcs as a parameter. The position of extra arcs is chosen randomly. If there already exists an arc between nodes of a Petri net, the weight for this arc is increased. The resultant Petri net may be unbounded and may contain deadlocks, livelocks and dead transitions.

At the last step, variables and conditions are added to the Petri net from the previous step. The number of variables and the number of atomic conditions are defined by a user. Currently, all variables and conditions are associated with the domain of real numbers, as it has been proven that the soundness verification algorithm always terminates for this class of nets. The given number of variables is generated and added to the net. Each variable is initialized to 0. To generate conditions, we first generate a set of constants for the conditions. Since in real nets the same constants may appear in different conditions, we explicitly define an upper bound for the number of constants – currently, it is equal to the number of conditions divided by four. Based on the generated variables, constants and predefined predicates (e.g. for reals, $\mathcal{P} = \{=, \neq, >, \geq, <, \leq\}$), atomic conditions are generated. The conditions can be of both variable-operator-constant and variable-operator-variable forms. Each transition may be associated with zero, one or multiple conditions. If a transition is associated with multiple conditions, these conditions are combined using conjunction and disjunction (the logical connective between each pair of conditions is chosen randomly). For each transition guard, we ensure that this guard is not identically false. When all the conditions are generated and added to the net, the resultant DPN is returned to a user.

Figure 1.1 illustrates the results of these steps. Step 1 corresponds to the generation of a sound backbone for a DPN. Step 2 corresponds to the addition of extra arcs to the net. Step 3 corresponds to the addition of variables and conditions to the net. The resultant DPN is unsound due to its unboundedness.

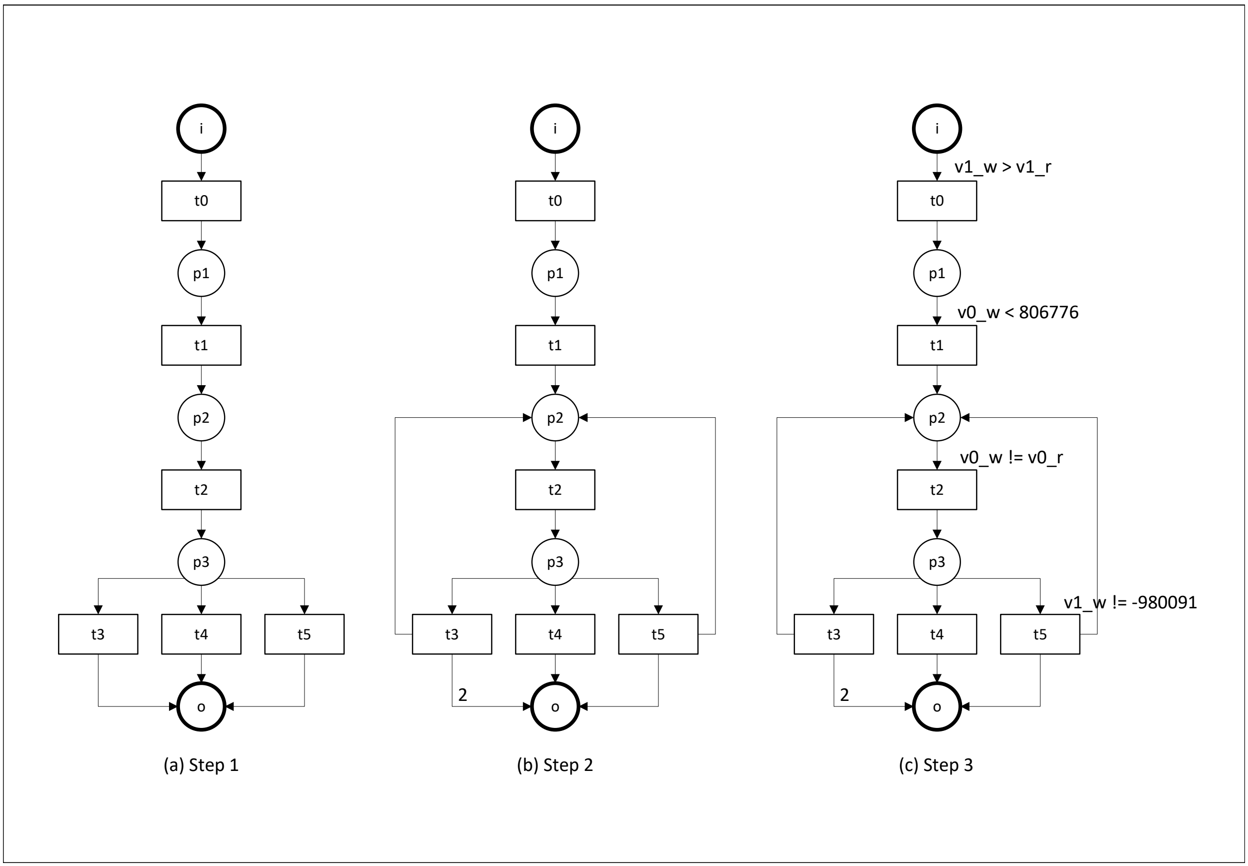


Figure 1.1: Steps to generate a DPN with 5 places, 6 transitions, 3 extra arcs, 2 variables and 4 conditions.

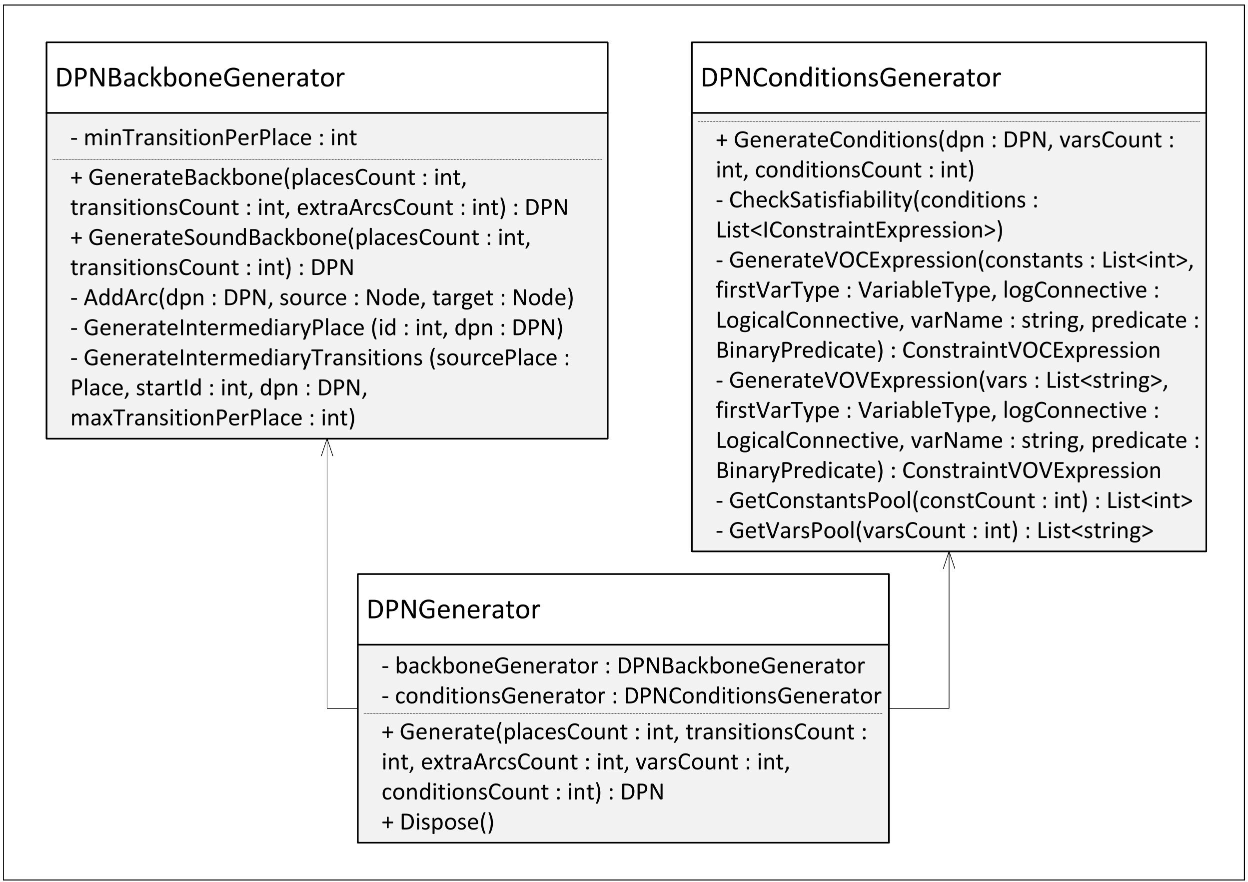


Figure 1.2: Class Diagram for the class library implementing the algorithm for DPN generation

The algorithm for generation of sample DPNs has been implemented as a .NET class library. The UML Class Diagram for this library is presented in Figure 1.2. There are three main classes: *DPNBackboneGenerator*, *DPNConditionsGenerator*, and *DPNGenerator*. *DPNBackboneGenerator* allows to generate a sound Petri net and to add arcs to this net. *DPNConditionsGenerator* allows to add variables and conditions to a given net. *DPNGenerator* allows to generate a DPN with the predefined parameters described above.

The example of generation of a sample DPN is presented in Figure 1.3.

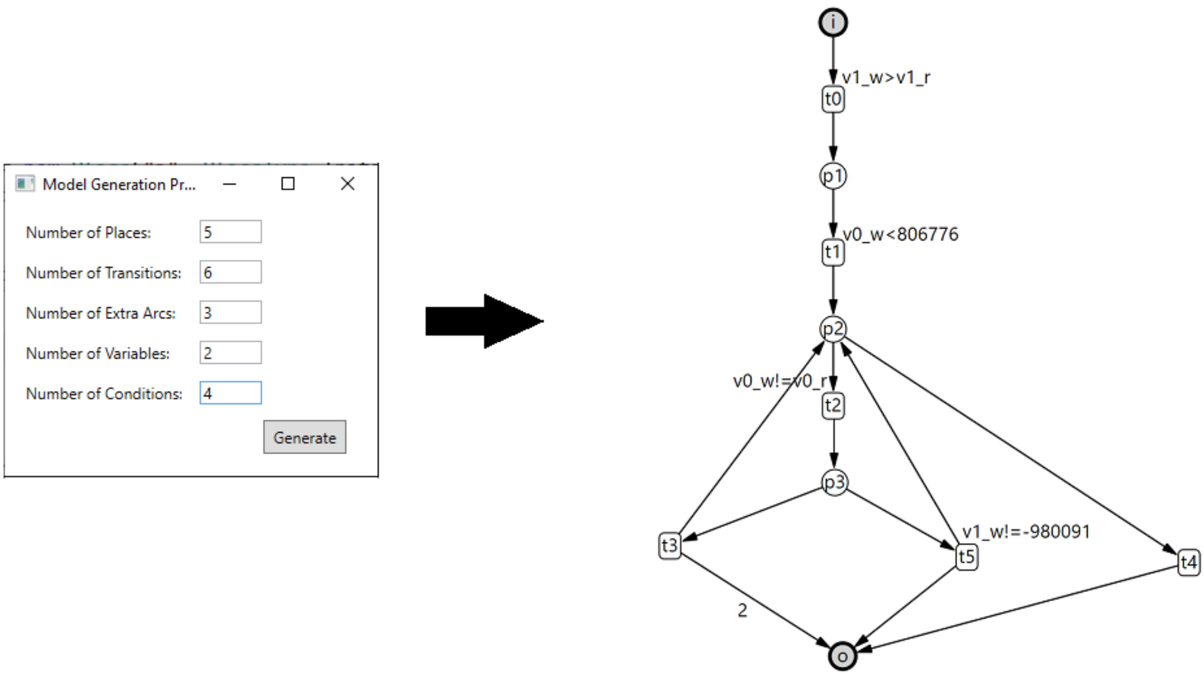


Figure 1.3: Example of using DPN generation in a research prototype

1.2 Iterative Generation and Verification of DPNs

To evaluate the soundness verification algorithm on a sufficient amount of process models, we have developed a toolkit that allows to automatically generate and verify process models of different sizes. There are 5 main subsystems, namely DPNIterativeVerificationRunner, DPNVerifierForGivenParameters, DPNGenerator, ConstraintGraph, and CGAnalyzer, that are used in iterative DPN generation and verification. The interaction between these modules during the process of iterative DPN generation and verification is presented in Figure 1.4.

DPNVerifierForGivenParameters generates and verifies DPNs that conform to the predefined input parameters. When verification is done, information about the time spent on verification is logged in a CSV-file. Output information that DPNVerifierForGivenParameters writes in a CSV-file has the following properties:

1. PlacesCount: the number of places in a DPN.
2. TransitionsCount: the number of transitions in a DPN.
3. ArcsCount: the number of arcs in a DPN.
4. VarsCount: the number of variables in a DPN.
5. ConditionsCount: the number of conditions in a DPN.
6. Boundedness: whether a DPN is bounded or not.
7. ConstraintStates: the number of states in a constraint graph.
8. ConstraintArcs: the number of arcs in a constraint graph.

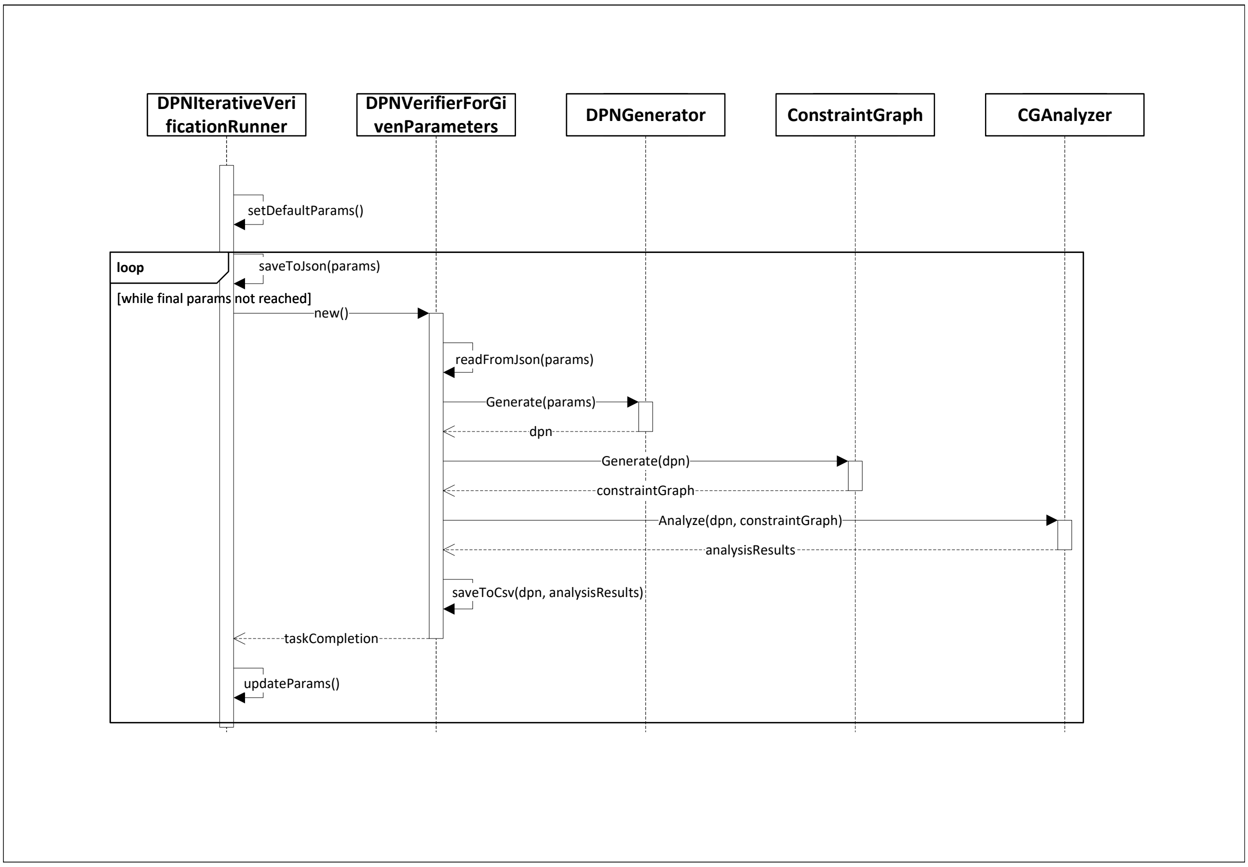


Figure 1.4: Sequence Diagram showing a process of iterative DPN generation and verification.

9. DeadTransitions: the number of dead transitions in a DPN.
10. Deadlocks: existence of deadlocks in a DPN.
11. Soundness: whether a DPN is data-aware sound.
12. Milliseconds: how much time the algorithm spent for verifying a DPN for soundness.

Both DPNVerifierForGivenParameters and DPNIterativeVerificationRunner are implemeted as .NET console applications, while other modules are implemented as class libraries. At each iteration of the loop from Figure 1.4, DPNIterativeVerificationRunner creates a new process for DPNVerifierForGivenParameters and waits until it exits. We have decided to run them in separate processes by the following reasons. Firstly, garbage collection does not always allow to clean all the unused data generated by Z3 and, thus, OutOfMemoryException may occur after executing DPNVerifierForGivenParameters for numerous times. Secondly, the execution of Z3 procedure can only be interrupted by the termination of the parent process. Therefore, if Z3 requires too much time to perform some action, there is no way to interrupt it but to terminate the parent process. Based on these reasons, it was decided to run DPNVerifierForGivenParameters in a separate process that may be killed and re-created when some waiting time limit is exceeded.

Figure 1.5 shows console output of DPNIterativeVerificationRunner.exe after some runs of DPNVerifierForGivenParameters. Figure 1.6 shows the verification results for these runs that are logged in a CSV-file.

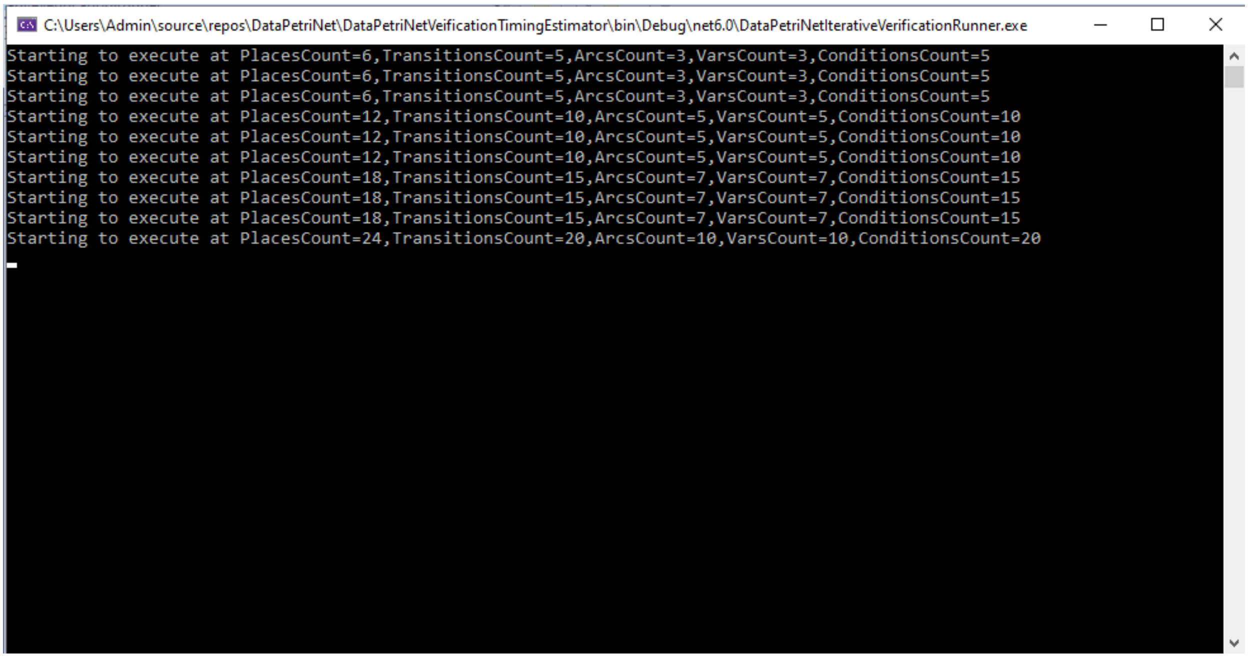


Figure 1.5: Output of DPNIterativeVerificationRunner application after several DPN generations and verifications.

```
6,5,12,3,5,True,13,15,1,True,False,91
6,5,13,3,5,False,6,5,1,False,False,38
6,5,12,3,5,True,42,76,0,False,False,230
6,5,12,3,5,True,4,3,2,True,False,13
6,5,12,3,5,True,4,3,2,True,False,20
6,5,13,3,5,True,28,42,0,True,False,978
12,10,27,5,10,False,79,141,2,False,False,707
12,10,27,5,10,False,9,8,4,False,False,21
12,10,26,5,10,False,44,43,0,False,True,278
18,15,41,7,15,False,14,13,8,False,False,169
18,15,41,7,15,False,23,22,4,False,False,162
18,15,41,7,15,False,24,23,6,False,False,167
```

Figure 1.6: Contents of CSV-file after several DPN generations and verifications. Order of columns corresponds to order of output properties described above.

2 Performance Evaluation of the Algorithm for Soundness Verification

In this chapter, we describe the results of the experiments conducted on sample DPNs that have been generated and verified for soundness with the help of the toolkit proposed in the previous chapter. Section 2.1 shows how the time needed for the algorithm to verify soundness grows with the increase in a DPN size and compares two variants of the soundness verification algorithm. Section 2.2 evaluates the dependence of the time needed for the algorithm to verify soundness on each DPN characteristic.

The experiments have been carried out on a PC with the following properties:

- Motherboard: HP 836B,
- RAM: DDR4 16Gb 1200MHz,
- CPU: Intel Core i7-7700HQ @ 2.80GHz,
- Disk: Samsung SSD 980 500Gb,
- OS: Windows 10 x64.

2.1 Comparison of the Algorithm Variants

In this section, we show how the time needed for the algorithm to verify soundness grows with the increase in a DPN size. In particular, we define a common setup for DPNs, and verify soundness of DPNs that conform to this setup. Using Python and data analysis libraries, such as Pandas and MatPlotLib, we analyze results of multiple iterations of soundness verification.

Given $n \in \mathbb{N}$, we consider DPNs of the following setup:

- $1.2n$ places,
- n transitions,
- $0.5n$ extra arcs (or nearly $3n$ arcs, in total),
- $0.5n$ variables, and
- n conditions.

During the experiments, we generate 3 DPNs with less than 60% of dead transitions for each n. We start with $n = 5$. After each iteration, we increase n by 5. Information about the performed soundness verification for a DPN is stored in a CSV-file.

In this study we consider two variants of the soundness verification algorithm, which differ in a way how Algorithm 1 is implemented. The first approach is based on the usage of the Quantifier Elimination (QE) tactic that is already implemented in Z3 and that allows to automatically remove existential quantifier of a formula and get all the necessary implications. However, the QE tactic in Z3 is implemented in such a way that the result expression may not be in a proper normal form; that is why, the expression returned by the quantifier elimination procedure has to be converted to a conjunctive normal form (CNF) using Tseytin transformation and, then, the result of this conversion has be to transformed to a DNF. The second approach overcomes the necessity of using these transformations by replacing the usage of the QE tactic with the usage of a non-standard procedure, which removes all the irrelevant conditions from a formula, generates all the implications and, unlike the QE-tactic, ensures that the resultant formula is always in a DNF. Although the second approach does not use the QE tactic, it has to perform additional optimization tasks and satisfiability checks to make implications, which slightly increases the computational complexity of the algorithm.

Figure 2.1 shows time needed for the algorithm variants to verify soundness of DPNs of different sizes. Abscissa represents n. Ordinate represents time (in ms) spent by the algorithm to determine soundness of the DPN of the given n. Although the time needed for the algorithm variants to verify

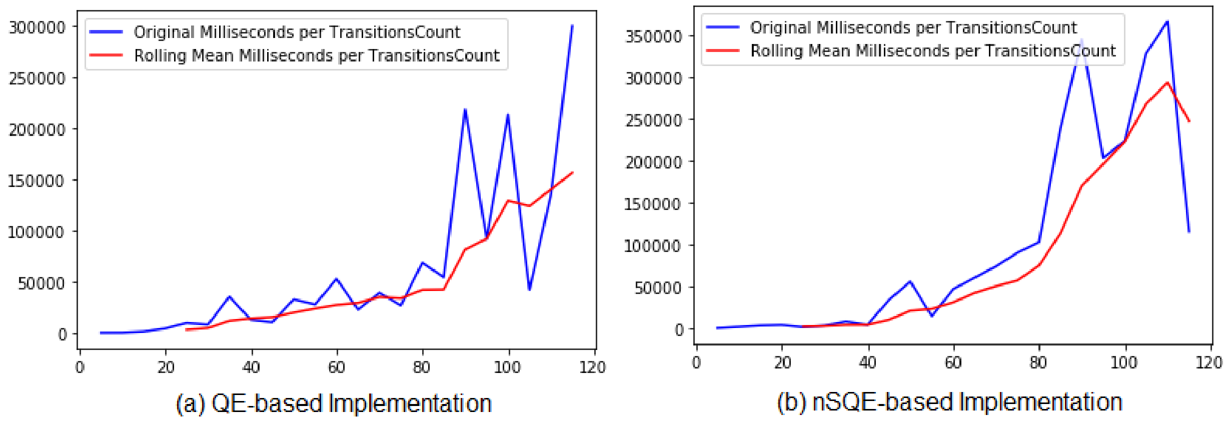


Figure 2.1: Time spent by the soundness verification algorithm variants for different n. (a) The variant that uses the QE-tactic to eliminate quantifiers. (b) The variant that uses the non-standard procedure to eliminate quantifiers.

soundness grows exponentially with an increase in n, the plots show that the proposed soundness verification algorithm variants are applicable on process models of both small and medium sizes. However, based on the plots, we can conclude that the verification time growth rate for the algorithm variant that uses the non-standard procedure to eliminate quantifiers is higher than the verification time growth rate for the algorithm variant that uses the QE tactic to eliminate quantifiers. It means that, in general, the QE-based variant allows to obtain the result of the soundness verification quicker. Due to this reason, in the next section we consider specifically the QE-based variant to estimate dependence of the soundness verification time on each DPN characteristic.

2.2 Evaluation of the dependence of the soundness verification time on DPN characteristics

In this section, we show how DPN characteristics influence the time needed for the algorithm to determine soundness of a DPN. Specifically, we generate DPNs of different sizes by gradually incrementing each input parameter, and verify each generated DPN for soundness. The initial configuration includes 2 places, 1 transition, 0 extra arcs, 1 variable, and 0 conditions. For each configuration, 10 DPNs are generated and verified for soundness. Verification information is saved in a CSV-file. After generation and verification stages, either one or all input parameters are increased by some constant. The final DPN configuration includes 255 places, 255 transitions, 255 extra arcs, 260 variables and 155 conditions. Table 2.1 demonstrates starting and ending configurations for different increment values. Initially, each parameter is increased only by 1, but with an increase in a DPN size, the increment value also increases, which is denoted by the table. Verification information that is saved in the CSV-file is then analyzed using Python and data analysis libraries. We first construct plots showing the dependence of the soundness verification time on each DPN characteristic. Then, we construct a regression model to determine factors whose influence on the time needed for the algorithm is significant and to estimate the impact of each factor on the resultant time needed for the algorithm.

The resultant dataset containing verification information consists of 36256 rows meaning that 36256 DPNs have been generated and analyzed. Some verifications have not been logged: we have interrupted verification if, during the execution of Algorithm 1, Z3 transformed some expression into a DNF expression that contains some disjunct with more than 75000 terms or if a transformation of a single expression block to a DNF required more than 15 seconds. Occurrence of such situations is mainly the consequence of the way how the transformation to DNF is performed. Initial expression is first transformed to a CNF using Tseytin transformation, and, then, the resultant

Table 2.1: Properties of DPN generation. $|P|$ denotes the number of places, $|T|$ denotes the number of transitions, $|A_{extra}|$ denotes the number of extra arcs, $|V|$ denotes the number of variables, $|\Phi_{atom}|$ denotes the number of atomic conditions in a DPN.

Increment	Configuration	$\|P\|$	$\|T\|$	$\|A_{extra}\|$	$\|V\|$	$\|\Phi_{atom}\|$
1	Starting Configuration	2	1	0	1	0
	Ending Configuration	25	25	25	25	25
3	Starting Configuration	27	26	10	5	5
	Ending Configuration	75	77	76	77	77
5	Starting Configuration	78	78	25	15	15
	Ending Configuration	125	125	125	150	110
15	Starting Configuration	135	135	75	35	35
	Ending Configuration	255	255	255	260	155

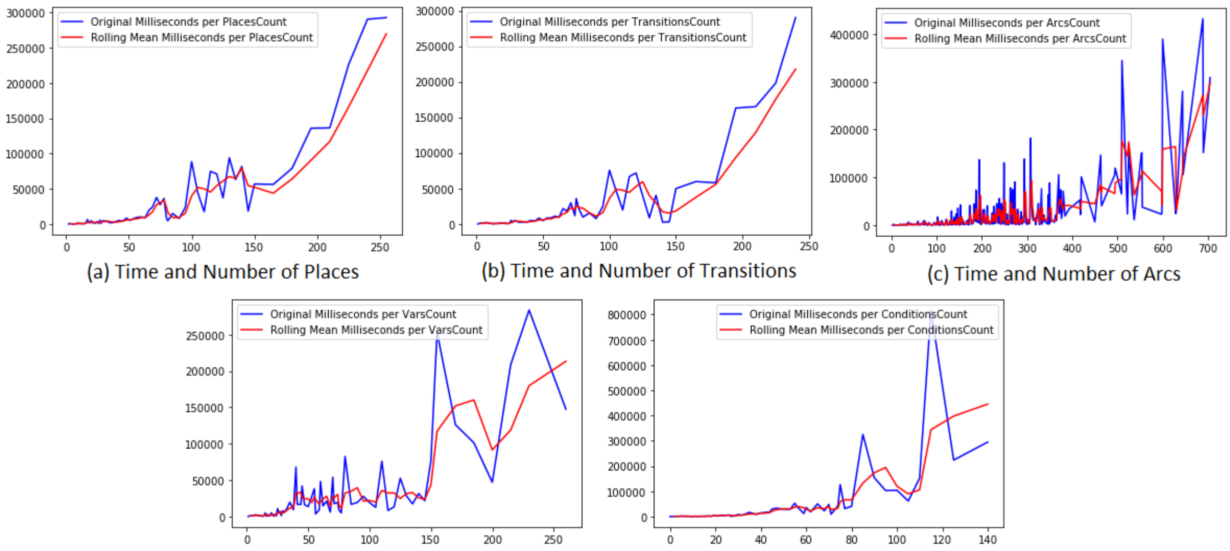
Table 2.2: Spearman correlations between DPN characteristics. Only DPNs with less than 60% of dead transitions are considered.

	$\|P\|$	$\|T\|$	$\|A\|$	$\|V\|$	$\|\Phi_{atom}\|$
$\|P\|$	1	0.87	0.95	0.8	0.82
$\|T\|$	0.87	1	0.97	0.76	0.77
$\|A\|$	0.94	0.97	1	0.76	0.78
$\|V\|$	0.8	0.76	0.76	1	0.75
$\|\Phi_{atom}\|$	0.82	0.78	0.78	0.75	1

CNF-expression is transformed into a DNF using distribution of terms. This approach may lead to an exponential blowup of the formula size. The complexity of quantifier elimination as well as transformation to DNF depends on the number of atomic formulas. That is why, if the number of terms in a formula exceeds some threshold, operations over this formula cannot be performed in a reasonable amount time. Thus, we decided to stop the algorithm execution if such situations appear. Notably, in the own implementation of Algorithm 1, the growth of the formula size in such cases seems to be generally lower than in the QE-based implementation since the conversions to CNF and DNF are omitted. Currently, we exclude these cases from the scope of our research. In future, we plan to deeply investigate cases when the algorithm cannot terminate in a reasonable amount of time and propose approaches that may help to overcome this limitation.

To analyze data, we consider only DPNs with less than 60% of dead transitions. There are 8526 cases out of 36256 that satisfy this condition. We first examine the dependence of the time spent on verification on each DPN characteristic. Figure 2.2 illustrates how an increase of places, transitions, arcs, variables and atomic conditions in a DPN influences the time needed for the algorithm to determine soundness of the DPN. Plots show that, in general, the time needed for the algorithm grows exponentially with an increase of each of these parameters. However, it can be explained by the way how the experiment was conducted (see Table 2.1), since there are sufficiently high correlations between DPN characteristics. Table 2.2 demonstrates Spearman correlation between each DPN characteristic of generated nets with less than 60% of dead transitions.

To estimate significance and influence of each parameter on the time needed for the algorithm,



(a) Time and Number of Places

(b) Time and Number of Transitions

(c) Time and Number of Arcs

(d) Time and Number of Variables

(e) Time and Number of Atomic Conditions

Figure 2.2: Dependence of soundness verification time on each of the DPN parameters. Abscissa represents a DPN parameter. Ordinate represents time (in ms) spent by the algorithm.

we construct a linear regression model. As a dependent variable, we take $ln(Milliseconds + 1)$ since Figure 2.2 shows an exponential relationship between each DPN characteristic and the time spent by the algorithm. Based on the experiments conducted, we have decided to replace the number of places and the number of transitions with the single variable $net\_size = |P| \times |T|$ to increase the quality of the model. Using Python library *statsmodel*, we construct a linear regression model for dependent variable $ln(Milliseconds + 1)$ and predictors net_size, $|A|$, $|V|$, $|\Phi_{atom}|$ using ordinary least squares (OLS) method. The results are illustrated in Figure 2.3.

```
                            OLS Regression Results
==============================================================================
Dep. Variable:       log_milliseconds   R-squared:                       0.638
Model:                            OLS   Adj. R-squared:                  0.638
Method:                 Least Squares   F-statistic:                     3762.
Date:                Fri, 15 Jul 2022   Prob (F-statistic):               0.00
Time:                        10:46:57   Log-Likelihood:                -15615.
No. Observations:                8526   AIC:                         3.124e+04
Df Residuals:                    8521   BIC:                         3.128e+04
Df Model:                           4
Covariance Type:            nonrobust
===================================================================================
                      coef    std err          t      P>|t|      [0.025      0.975]
-----------------------------------------------------------------------------------
const               2.9498      0.027    108.491      0.000       2.897       3.003
net_size           -0.0003   8.29e-06    -32.140      0.000      -0.000      -0.000
ArcsCount           0.0187      0.001     37.014      0.000       0.018       0.020
VarsCount           0.0170      0.001     16.653      0.000       0.015       0.019
ConditionsCount     0.0706      0.002     42.435      0.000       0.067       0.074
==============================================================================
Omnibus:                     1347.816   Durbin-Watson:                   1.173
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             2527.285
Skew:                           0.993   Prob(JB):                         0.00
Kurtosis:                       4.781   Cond. No.                     7.90e+03
==============================================================================
```

Figure 2.3: Summary of regression results.

The results show that the model is statistically significant at $p = 0.99$. Each predictor is statistically significant at $p = 0.99$. To estimate influence of each predictor on the dependent variable, we compute $(e^{coef} - 1) * 100\%$. The obtained results can be interpreted as follows:

- An increase in $|P| \times |T|$ by 1, on average, leads to a decrease of the time spent by the algorithm by 0.02%.
- An increase in $|A|$ by 1, on average, leads to an increase of the time spent by the algorithm by 1.9%.
- An increase in $|V|$ by 1, on average, leads to an increase of the time spent by the algorithm by 1.7%.
- An increase in $|\Phi_{atom}|$ by 1, on average, leads to an increase of the time spent by the algorithm by 7.3%.

Based on the results presented above, we can conclude that the number of atomic conditions is a parameter that influences the verification time the most. An addition of one atomic condition to a DPN, on average, increases the time needed for the algorithm to determine soundness of the model by 7.3%. The conducted experiments also showed that in most cases the soundness verification algorithm cannot terminate in a reasonable amount of time if the number of atomic conditions is greater than 140 and the amount of dead transitions is less than 60%. However, such cases must be investigated more deeply. Potentially, the algorithm can terminate in a reasonable amount of time if each transition guard is an atomic condition or a conjunction of write-conditions: in this case, we can avoid disjunctions in constraint state formulas and, by that, avoid exponential blowup in a formula size. In future, we plan to check this hypothesis. We also plan to compare time needed to verify soundness of DPN with and without guards to examine how much the verification time increases when transition guards are added.